Maximierung der kooperativ sichtbaren Fläche durch eine Gruppe humanoider Roboter

Bachelorarbeit

Alexander Stöwing

Matrikelnummer: 2787395

16. November 2015

Fachbereich Mathematik / Informatik
Studiengang Informatik

1. Gutachter: Dr. Tim Laue
2. Gutachter: Prof. Dr. Rolf Drechsler

Erklärung

Ich versichere, den Bachelor-Report oder den von mir zu verantwortenden Teil einer Gruppenarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, den 28. Februar 2020

.....

(Alexander Stöwing)

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dies ist eine Arbeit die im Rahmen des studentischen Projekts B-Human entstand. B-Human ist ein Team der Universität Bremen in Kooperation mit dem Deutschen Forschungszentrum für künstliche Intelligenz, welches regelmäßig an den internationalen Wettbewerben des RoboCups ([13]) in der Standard Plattform Liga (SPL [14]) teilnimmt und dabei in den Jahren 2009, 2010, 2011 und 2013 den Weltmeistertitel gewinnen konnte. In der SPL spielen *NAO*-Roboter (vgl. Kapitel 2.3) in 5-er Teams gegeneinander Fußball.

1.1 Motivation

Im Rahmen des Projekts B-Human sollen die Studierenden Software für den humanoiden Roboter *NAO* entwickeln. Das Projekt existiert schon seit einiger Zeit und ist somit über die Jahre gewachsen. Daher gibt es zwar einige noch offene Punkte, aber auch Module, die es erneut zu prüfen und möglicherweise mit neuen Lösungsansätzen zu ergänzen gilt. Die Kopfsteuerung ist ein altes Modul, welches von keinem der aktuellen Projektmitglieder entwickelt wurde. Daher wäre eine Neuentwicklung durchaus sinnvoll damit es wieder eine Expertise innerhalb des Teams gibt. Da ich schon mehrere kleinere Verbesserungen an der Kopfsteuerung vorgenommen habe sah ich mich in der Verantwortung dieses Modul zu überarbeiten und wollte dies als meine Bachelorarbeit in Angriff nehmen. Mein persönliches Ziel war es ein leicht modifizierbares Modul zu schreiben, was einen neuen logischen Ansatz

verfolgt. Dieser logische Ansatz ist die kooperative Flächenabdeckung des Spielfeldes, welche in dieser Arbeit untersucht wird. Die Motivation besteht somit darin einen geometrischen Ansatz zur Kopfsteuerung auf humanoiden Robotern auszuprobieren und dadurch das Spielverhalten zu verbessern.

1.2 Ziele und Aufbau der Arbeit

Das Ziel dieser Arbeit ist wie oben beschrieben eine neue Kopfsteuerung zu entwickeln und in ihrer Anwendung zu untersuchen. Der bisherige Ansatz ist es mit einem Roboter nach Möglichkeit interessante Feldpunkte anzusteuern oder einfach seinen Blickwinkel etwas zu verändern. Die hier vorgestellte Entwicklung verfolgt zusätzlich einen anderen Ansatz in dem man möglichst viel von dem Feld über die Sichtfelder der Roboter abdeckt. Um dies zu erreichen werden viele geometrische Operationen benutzt, die in der Literatur als sehr aufwändig gelten. Um diese Arbeit näher zu erklären werden in dem nächsten Kapitel der Hintergrund, der RoboCup und die Hardware beschrieben. Im Anschluss werden verwandte Arbeiten im Kontext dieser Arbeit untersucht. Danach werden die Grundlagen erläutert, die hinter der Software stecken. Dabei werden sowohl Prinzipien, Algorithmen als auch Datenstrukturen angeschnitten. In den darauffolgenden Kapiteln wird Schrittweise erklärt wie die Maximierung der sichtbaren Fläche erfolgt und anschließend wird der genaue Mechanismus der Kopfsteuerung erklärt. Als letztes folgen die Evaluation und der Ausblick, die sich kritisch mit der hier gebrachten Leistung auseinandersetzen und ein Fazit ziehen.

Kapitel 2

Hintergrund

In diesem Kapitel wird ein kurzer Überblick über den RoboCup, den *NAO* und das Team B-Human gegeben.

2.1 RoboCup

Der "RoboCup" ist eine internationale wissenschaftliche Initiative, die es sich zum Ziel gesetzt hat die Entwicklung intelligenter Roboter voranzutreiben. Als offizielles Ziel bei der Gründung 1997 setzte man sich auch dem amtierenden *FIFA* Fußballweltmeister im Jahr 2050 mit einer Gruppe autonomer humanoider Roboter auf dem Feld zu begegnen und zu gewinnen.
Im Rahmen dieser Initiative sind mehrere Themengebiete hinzugekommen. Insgesamt gibt es die Gebiete *RoboCupSoccer*, *RoboCupRescue*, *RoboCup@Home* und *RoboCupJunior* [13].
Jährlich wird eine Weltmeisterschaft ausgetragen und kleinere Veranstaltungen des Wettbewerbs werden auch durchgeführt, wie zum Beispiel die *GermanOpen*. Ein Ausschnitt eines Spiels ist auf Abb. 2.1 zu sehen.

2.2 B-Human

Das Team B-Human ist eines der besten Teams in der RoboCup Standard Plattform Liga (SPL) und hatte in den Jahren 2009, 2010, 2011 und 2013 den Weltmeistertitel erringen können. In dem Jahr 2015 wurde B-Human

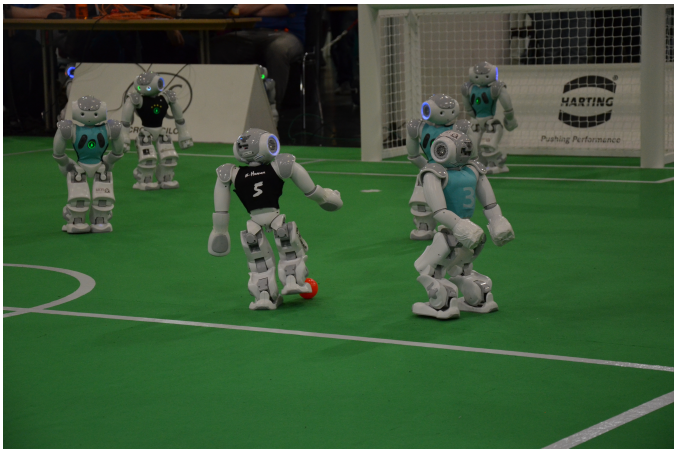


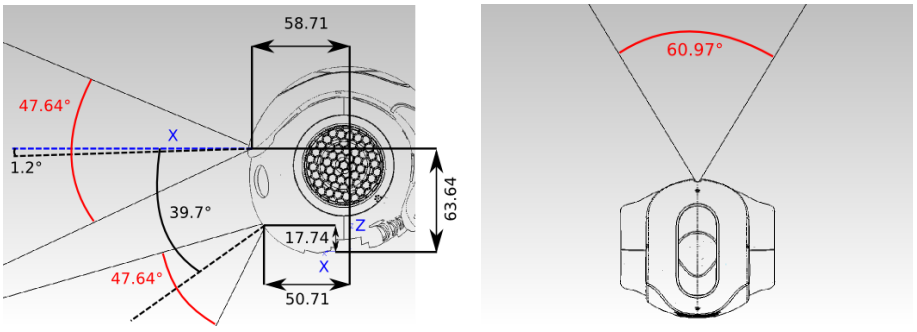
Abbildung 2.1 B-Human bei einem Spiel auf der GermanOpen 2015.

Vizeweltmeister. Seit 2009 ist B-Human in der GermanOpen ungeschlagen [19]. Das besondere an dieser Liga ist, dass es ein reiner Software Wettbewerb ist, da als Standard Plattform der *NAO* verwendet wird und dieser nicht modifiziert werden darf [14].

2.3 Der Nao

Der *NAO* ist ein 58 cm großer programmierbarer humanoider Roboter von der französischen Firma Aldebaran Robotics, der 25 Freiheitsgrade besitzt und durch elektrische Motoren und Aktuatoren gesteuert wird. An Sensorik besitzt dieser Roboter zwei Kameras, vier direktionale Mikrophone, Sonar Entfernungsmesser, zwei IR Sender und Empfänger, ein Intertial Board, neun Tastsensoren und acht Drucksensoren. Als Prozessor besitzt der *NAO* einen Intel ATOM 1.6 GHz CPU im Kopf der mit einem Linux Kernel betrieben wird und die Aldebaran eigene proprietäre Middleware unterstützt. Sein äußeres Erscheinungsbild ist auf Abb. 2.2 zu sehen.
Zu den Gelenken ist noch wissenswert dass diese die Kopfbewegung ermöglichen so wie die Bewegung der Extremitäten, wobei die Beine durch einen gemeinsamen Motor in der Hüfte gesteuert werden.
Die Kameras des *NAO* sind zwei 1280×960 Kameras. Wobei die eine auf der Stirn des Roboters angebracht ist und die andere auf Höhe des Munds [16], wie in Abb. 2.3 dargestellt.

Abbildung 2.2 Der *NAO*. [17]



(a) Seitenansicht (b) Ansicht von Oben

Abbildung 2.3 Die Kameras des *NAO* mit eingezeichneter Blickrichtung samt Öffnungswinkel. [18]

Kapitel 3

Verwandte Arbeiten

Es gibt viele Arbeiten die sich mit der Steuerung von Robotern beschäftigen und deren Ziele können sehr divers sind.
Eine interessante Arbeit zum Thema Kopfsteuerung ist die Arbeit von Andreas Seekircher [6]. Hier ist vor allem eine Verbesserung des Weltmodells das Ziel in dem man durch das aktive „Wahrnehmen“ der Umgebung, unter Zuhilfenahme des aktuellen Weltmodells, die Entropie einer unterliegenden Partikelverteilung minimiert. Diese Arbeit basiert auf solider theoretischer Grundlage und wird Anhand der Lokalisierung und der Ballerkennung getestet. Eine Gemeinsamkeit zwischen der Arbeit von Seekircher und dieser hier ist der Fokus darauf rechenintensive Methoden echtzeitfähig umzusetzen [6].
Ähnliche Vorgehensweisen gibt es auch in anderen Arbeiten bei denen man nicht allgemein das aktive „Wahrnehmen“ durchführt, sondern die Kopfsteuerung von der gerade auszuführenden Aktion abhängig macht, ähnlich wie bei Menschen. Dabei muss man einen Ausgleich zwischen der zielgesteuerten Observation und der Lokalisierung finden. In der Arbeit von Guerrero et al. hat man eine Kopfsteuerung entwickelt, welche die Torabdeckung für den Torwart verbessert. Dabei wurden Kalman-Filter mit Gaußscher Darstellung und Partikelfilter verwendet [7]. Diese Arbeit geht auch davon aus das die Wahrnehmung aktiv erfolgt, wie in der Arbeit von Seekircher. Die Roboter müssen aber in der Arbeit von Guerrero et al. alle Entscheidungen selbstständig treffen, ebenso auch die Wichtigkeit der Ziele festlegen, die in dieser Arbeit in gewissem Rahmen vorgegeben ist.
Bei der ganzen Thematik sind Lernalgorithmen sehr beliebt um signifikant

den Aufwand der Ausführung zu reduzieren und dabei zu helfen die Umgebung einzuschätzen.
Diese Arbeiten haben mit dieser vor allem die Hardware gemein, da alle mit dem *NAO* arbeiten und daher auf seine Möglichkeiten beschränkt sind.
Eine Arbeit die nicht dieser Beschränkung unterliegt ist die von Lynn E. Parker. Auch hier wird eine Sensorsteuerung behandelt und einige ihrer Probleme thematisiert, darunter wird auch die Komplexität vorheriger geometrischer Ansätze erwähnt. Daher wird auf eine weniger komplexe Herangehensweise gesetzt. Es wird eine online Kontrollverteilungsstrategie, in diesem Kontext entspricht dieser Begriff einer Kopfsteuerung, entwickelt, um die Zeit in der möglichst viele Ziele beobachtet werden zu maximieren. Dieses Problem wird formalisiert und als *CMOMMT - Cooperative Multi-Robot Observation of Multiple Moving Targets* bezeichnet. Als Lösung werden lokale Kräftevektoren benutzt, die nach einer Heuristik verfahren. Dabei bietet sich auch diese Heuristik zum Lernen an. Ein wesentlicher Unterschied ist hier die Hardware, da die Roboter eine 360 Grad Sensorabdeckung haben. Daher ist die Aufgabenstellung eine Sache der gesamten Roboterbewegung und nicht der bloßen Kopfsteuerung, auch wenn sich hier einige Prinzipien adaptieren lassen [1]. Eine Ähnlichkeit besteht aber schon da eine Überschneidung des Sensorraums der Roboter durch die Heuristik zur optimalen Zielverfolgung eher negativ ist. In dieser Arbeit wurde die Heuristik entsprechend verändert. So versuche ich hier auch eine Überschneidung des Sensorraums (der Sichtfelder) zu vermeiden.
Die Arbeit von Seara et al. ist aus der Biologie inspiriert und möchte die sichtbaren Informationen maximieren. Dazu wird ein stochastisches Modell des Roboters und des Wahrnehmungssystems benötigt um die ganzen Abhängigkeiten zu modellieren und die Zustandsschätzung mit einem Kalman-Filter zu ermöglichen. Als der Garant zum erfolgreichen Ausführen einer Aufgabe wird hier, wie in der Arbeit von Guerrero, der visuelle Fokus auf diese Aufgabe genannt mit dem man die visuellen Informationen für diese Aufgabe auch maximiert. Diese Kopfsteuerung ist darauf ausgerichtet, dass ein zweibeiniger Roboter durch die Wahrscheinlichkeitsverteilungen entscheidet, wo er hingucken möchte um seine Aufgabe des Hindernislaufs zu meistern [8].
Eine andere Arbeit die sich mit Kopfsteuerungen beschäftigt ist die Arbeit von Kwok und Fox. Ihr Ziel ist es eine Steuerung durch Reinforcement Learning zu erreichen. Dabei verwendet man Least Squares Reinforcement Learning, um die Ziele zu lernen, die für die Aufgabe relevant sind. Insgesamt

nutzt man Least Squares Policy Iterations und den Markov Decision Process um für den AIBO Roboterhund einen Ausgleich zwischen Lokalisierung und der Kopfsteuerung für den Torschuss zu finden [9].
Einige Funktionen dieser Arbeit wurden schon im Rahmen von B-Human benutzt. So stammt die Sichtfeldberechnung aus dem Ansatz der Feldabdeckung aus [10]. Dieser sollte den Robotern auf eine kooperative Weise eine effiziente Ballsuche ermöglichen, welche nicht nur auf das bloße Laufen mit einer ständigen Kopfbewegung setzt. Die Roboter kommunizieren dabei welche Teile des Feldes sie im Blick haben und können organisiert nach dem Ball suchen. Um diese Suche auf geeignete Weise zu modellieren wird ein Zellengitter über das Feld gelegt. Dabei werden alle Zellen deren Mittelpunkte in einem Sichtfeld liegen als potentiell gesehen markiert. Diese Zellen werden dann als sichtbar bezeichnet, wenn diese nicht von einem anderen Roboter verdeckt werden und auch nicht weiter als 2 Meter von dem Roboter entfernt sind. Mit diesen Bedingungen verhindert man, dass der Roboter seine Erkennungsfähigkeiten überschätzt. Jede dieser sichtbaren Zellen bekommt einen Zeitstempel, um die schon abgearbeiteten Zellen zu modellieren und mit diesem Wissen über die am wenigsten beobachteten Zellen eine Steuerung vorzunehmen. Dabei wird das aus dem Feld schießen des Balles als ein Neustart des Zellengitters behandelt, wobei die Zelle interessant wird die am längsten nicht mehr gesehen wurde. Damit die Roboter berechnen können wo sie den Ball zu suchen haben, werden die sichtbaren Zellen kommuniziert. Um das Wissen nun zusammenzubringen, erhält jede Zelle die von allen Roboter kommunizierte größte Abdeckung. Um den Schwellwert zur Unterscheidung von sichtbaren und nicht sichtbaren Zellen zu ermitteln, wird der Otsu Algorithmus verwendet. Die anschließend mit dem Schwellwert ermittelten nicht sichtbaren Zellen werden mit dem k-Means-Algorithmus auf die k Roboter verteilt. Dabei sind diese k Roboter, nur jene Roboter die stehen und sich in ihrer Lokalisierung recht sicher sind. Diese nicht sichtbaren Zellen werden dann mit dem four-way flood fill Algorithmus verbunden und aus dem resultierendem Gebilde wird dann der Mittelpunkt berechnet. Dieser Mittelpunkt wird von dem Roboter angesteuert und kommuniziert, damit sich die Wege der Roboter nicht kreuzen.

Da durch das definierte Feld in der Standard Plattform Liga nach dem Regelbuch[12] eine berechenbare Umgebung für den *NAO* geschaffen wurde,

bieten sich von daher geometrische Verfahren an. Diese Arbeit möchte dieses Wissen nutzen und folgt dabei dem Ansatz die kooperativ sichtbare Fläche zu maximieren. Dabei ist nicht primär das Ziel die Lokalisierung zu verbessern, sondern eher eine effiziente Ereigniserkennung. Dabei wird auf die Kommunikation gesetzt. Da nach diesem Ansatz möglichst nur ein Roboter das Ereignis wahrnimmt kann dies aber auch zu spezifischen Problemen führen, da diese Steuerung nicht immer die intelligenteste Aktion ist, da die Roboter im Verlauf eines Spiels die unterschiedlichsten Rollen inne haben und somit auch unterschiedliche Ziele haben. Daher wird in der Kopfsteuerung ein Kompromiss eingegangen, der im Kapitel Kopfsteuerung (vgl. Kapitel 7) erklärt wird.

Kapitel 4

Grundlagen

Dieses Kapitel behandelt einige Grundlagen, die für das vollständige Verständnis der nachfolgenden Kapitel essentiell sind.

4.1 Die Lochkamera und ihre Projektion

Die Bildaufnahme auf dem *NAO* arbeitet nach dem Prinzip der Lochkamera. Eine Lochkamera ist ein geschlossener Raum mit einer sehr kleinen Öffnung an der Vorderseite und besitzt an der gegenüberliegenden Rückseite eine Bildebene. Durch diese Konstruktion werden Lichtstrahlen, die von einem Objekt ausgehend durch die Öffnung einfallen, geradlinig auf die Bildebene projiziert, wodurch ein verkleinertes und seitenverkehrtes Abbild der sichtbaren Szene entsteht. Wie in der Abb. 4.1 dargestellt läuft die optische Achse durch die Lochöffnung rechtwinklig zur Bildebene. Wenn man annimmt dass ein sichtbarer Objektpunkt in einer Distanz von Z von der Lochebene und im vertikalen Abstand Y über der optischen Achse befindet kann man folgende Berechnungen aufstellen:
Die Höhe der zugehörigen Projektion y wird durch zwei Parameter bestimmt, die feste Tiefe der Kamerabox f und den Abstand Z des Objekts vom Koordinatenursprung. Durch den Vergleich der ähnlichen Dreiecke ergibt sich der Zusammenhang proportional zur Tiefe der Box:

$$y = -f\frac{Y}{Z} \tag{4.1}$$

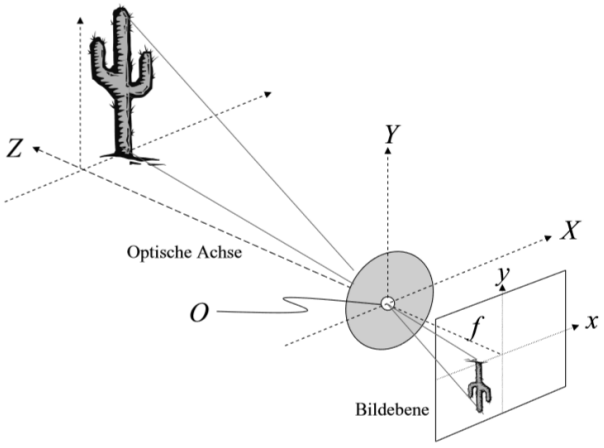


Abbildung 4.1 Visualisierung der Geometrie bezüglich der Lochkamera.[5], S. 7.

Analog verhält es sich bei der Breite, wobei x die Breite der Projektion ist und X der horizontale Abstand(vgl. [5], S.5-8):

$$x = -f\frac{X}{Z} \tag{4.2}$$

4.2 Rot-Schwarz-Bäume

Ein Rot-Schwarz-Baum ist ein 2-3-4-Baum der als binärer Suchbaum (engl. Binary Search Tree(BST)) implementiert wird. Dabei ist es hilfreich zu wissen, dass ein 2-3-4-Baum ein Suchbaum ist bei dem ein Knoten bis zu 3 Elemente beinhalten kann. Dabei hat ein solcher Knoten immer eine Kante mehr als er Einträge hat(vgl. [3], S.562).
Die Definition eines 2-3-4-Baumes klingt sehr unintuitiv und ist ebenso kompliziert zu implementieren. Daher gibt es die Datenstruktur des Rot-Schwarz-Baumes. Dieser realisiert das 2-3-4-Baum-Verhalten mit Knoten die nur zwei Kanten besitzen, aber dafür ein zusätzliches Farbbit haben, was die Funktionalität ermöglicht. Dabei ist egal ob dieses Bit in den Kanten gespeichert wird oder in den Knoten auf denen die Kanten zeigen. Für die Implementierung ist es einfacher dieses Bit in den Knoten zu speichern. Zur Verdeutlichung dient die Abb. 4.2.
Der Nutzen in den Rot-Schwarz-Bäumen liegt darin, dass er ein binärer Suchbaum ist und dadurch viele seiner Operationen aus dieser häufig verwendeten

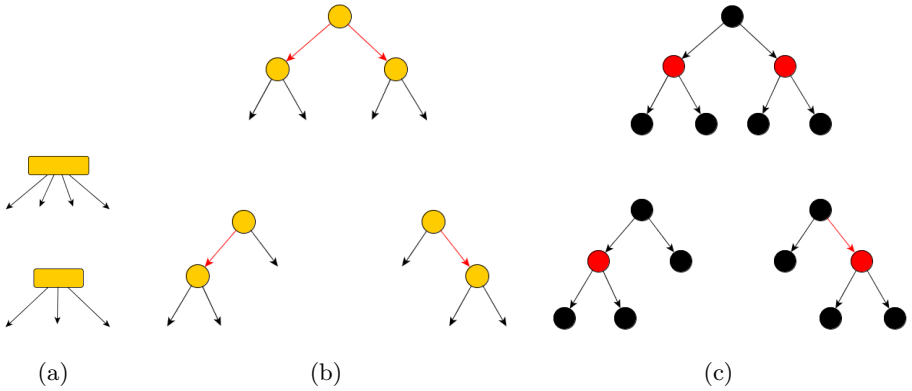


Abbildung 4.2 In (a) sind die interessanten Knoten eines 2-3-4-Baumes zu sehen.
(b) Die selben Knoten wie in Bild (a) bloß im Rot-Schwarz-Prinzip.
(c) Die Knoten wie in dem Bild (b) aus Sicht der Implementierung.
Angelehnt an [3], S.568.

Datenstruktur bezieht. Einige Operationen davon müssen dafür nur ein wenig erweitert werden. Die grundlegende Operation die zur Korrektheit und zum Ausbalancieren eines Rot-Schwarz-Baumes dient ist das Rotieren. Insgesamt definiert man Rot-Schwarz-Bäume folgendermaßen:

Rot-Schwarz-BST: Ein Rot-Schwarz-BST ist ein binärer Suchbaum, in dem jeder Knoten entweder Rot oder Schwarz markiert ist, wobei die zusätzliche Einschränkung gilt, dass keine zwei rote Knoten auf einem beliebigen Pfad von einer externen Verbindung zur Wurzel aufeinander folgen dürfen [3], S.575.

Ausgeglichener Rot-Schwarz-BST: Ein ausgeglichener Rot-Schwarz-BST ist ein Rot-Schwarz-BST, in dem alle Pfade von externen Verbindungen zur Wurzel die gleiche Anzahl von schwarzen Knoten enthalten [3], S.576.

Für die später vorgestellten mathematischen Operationen sind sortierte Listen interessant, bei denen man schnell bestimmte Objekte finden, einfügen und löschen kann. Daher werden in der Implementierung ausgeglichene Rot-Schwarz-Bäume benutzt, die auf dem Buch “Algorithmen in C++“[3] basieren, aber teilweise erweitert wurden.

Code-Abschnitte die aus dem Buch entnommen wurden sind:

1. Das Suchbaumgrundgerüst S.516.

2. Die Rotationen in einem binären Suchbaum S.532.
3. Das Zerlegen eines binären Suchbaums S.537.
4. Das Entfernen eines Knotens mit einem bestimmten Schlüssel aus einem binärem Suchbaum S.538.
5. Das Einfügen in einen Rot-Schwarz-Baum S.572.

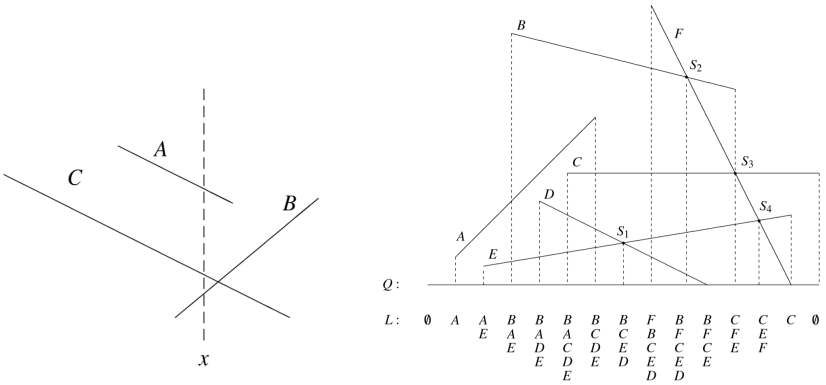
Alle implementierten Rot-Schwarz-Bäume vermeiden das doppelte Einfügen eines Elements in einen Baum und werden im Rahmen der Schnittpunktberechnung benötigt.

4.3 Das allgemeine Liniensegment-Schnittproblem

Das allgemeine Liniensegment-Schnittproblem soll zwei Probleme lösen.

1. Schnittpunkttest: Gibt es in einer Menge von n Segmenten mindestens ein Paar was sich schneidet?
2. Schnittpunktaufzählung: Bestimmung aller Paare sich schneidender Segmente in einer Menge von n Segmenten.

Für unsere Zwecke ist vor allem der zweite Punkt von Interesse. Da wir von einer Menge von Segmenten alle Schnittpunkte mit ihren beteiligten Segmenten ermitteln wollen und sich der erste Punkt damit beantworten lässt in dem man überprüft ob die resultierende Menge leer ist. Ein Ansatz wäre über die gesamte Menge der Segmente zu laufen und dabei jedes Segment mit jedem anderen Segment auf einen Schnitt zu überprüfen. Doch dieser Ansatz ist sehr aufwändig und gerade auf einem System was in Echtzeit arbeitet nicht empfehlenswert.
Um die Zahl der Schnittüberprüfungen zu minimieren, kann man das Prinzip der Scanline nutzen. Wir ziehen eine vertikale Scanline horizontal über die Ebene. Der Status der Scanline wird über die Menge der geschnittenen Segmente definiert, wobei dies eine lokale Ordnung ergibt die von der Höhe des Schnittes abhängt [4]. Beides wird in der Abb. 4.3 dargestellt.



(a) (b)

Abbildung 4.3 In (a) wird zur Vernschaulichung der Status einer Scanline auf einer x-Position dargestellt.[4], S. 487.
(b) Das Prinzip der Scanline auf einer Ebene mit Segmenten angewandt.[4], S. 491.

Für das gesamte Verfahren wird eine Eventqueue Q und eine Statusqueue L oder T, je nach Literatur, verwendet. Die Eventqueue beinhaltet dabei alle Punkte die während des Verfahrens besucht werden und ist dabei so sortiert, dass die Punkte mit den kleinsten x-Werten zuerst kommen. Haben zwei Punkte identische x-Werte kommt der Punkt mit dem kleineren y-Wert zuerst. Es wird während des Verfahrens immer der kleinste Werte aus der Eventqueue behandelt und dadurch werden weitere Mengen bestimmt, wie zum Beispiel die Menge der Segmente, die diesen Punkt als linken Endpunkt haben, die Segmente, die diesen Punkt als rechten Endpunkt haben und die Menge der Segmente, die den x-Wert innerhalb ihrer Spanne besitzen. Ebenso ist auch die Statusqueue wie oben beschrieben von dem gerade betrachteten Punkt abhängig. Damit man weniger auf Schnitte überprüfen muss wird darauf gesetzt die entstehenden Mengen zu überprüfen. Wenn zum Beispiel die Menge der Segmente, die diesen Punkt als linken oder rechten Endpunkt besitzen, größer als 2 ist gibt es einen Schnitt (vgl. [15], S.6 - 18).
Obwohl die Implementierung, wie sie vollständig in dem Kapitel 5 erklärt wird, viele der Prinzipien aus der Vorlesung [15] zu nutze macht, wie zum Beispiel die Verwendung von Bäumen als Datenstruktur und der unterschiedlichen Mengen, die aus einem Punkt ermittelt werden, so wird doch die Scan-

Richtung aus dem Lehrbuch [4] verwendet, wie in Abb. 4.4 dargestellt.

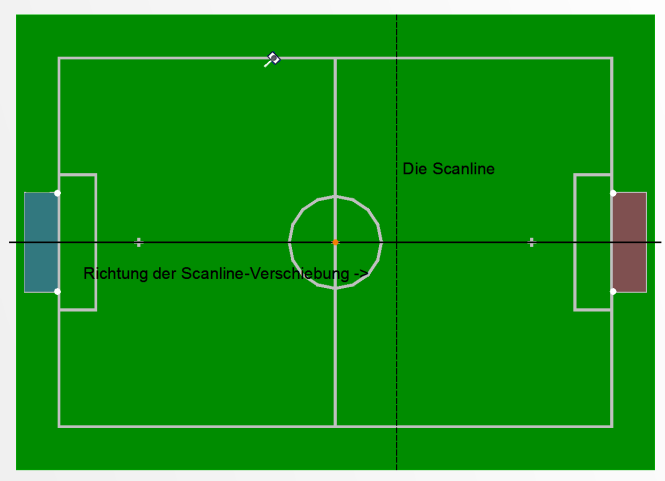


Abbildung 4.4 Verwendung der Scanline.

4.4 Die Gaußsche Trapezformel

Im Rahmen der Berechnung für die Feldabdeckung müssen die Flächen verschiedenster geometrischer Figuren berechnet werden. Beispiele sind das Feld (Rechteck) und die Sichtfelder (Trapeze). Wenn man davon ausgeht dass die Trapeze immer gleich angeordnet sind, was der Fall ist, kann man deren Fläche ebenso gut berechnen wie die Fläche des Feldes. Nicht mehr so trivial ist es die Schnittfläche von zwei Trapezen zu berechnen. Je nach Szenario können die unterschiedlichsten Formen mit ebenso unterschiedlichen Orientierungen herauskommen. Daher würde eine allgemeine Formel zur Berechnung der Flächeninhalte für unterschiedliche Polygone sehr nützlich sein. Im Rahmen der Implementierung wird hierfür die Gaußsche Trapezformel in der folgenden Form benutzt:

$$\mathbb{A} = \frac{1}{2} \cdot \sum_{i=1}^{n} (y_i + y_{i+1}) \cdot (x_i - x_{i+1}) \tag{4.3}$$

Diese Formel arbeitet mit der Eingabe von Eckpunkten der Form $A_1 = (x_1, y_1), A_2 = (x_2, y_2), \ldots, A_n = (x_n, y_n)$, dabei sind diese derart sortiert, dass die Punkte den Rand des Polygons in eine Richtung ablaufen. Es gilt bei der Anwendung der Formel $A_{n+1} = A_1$. Die Flächenberechnung erfolgt

in dem das Polygon in mehrere Trapeze zwischen den Punkten zerlegt wird. So sind nach der Formel die Flächen der Trapeze, die durch das Ablaufen in die eine Richtung entstehen positiv und die Anderen negativ, wie in Abb. 4.5 dargestellt. Durch die Aufsummierung erhält man letztendlich die Fläche des Polygons (vgl. [2], S. 1 - 5).

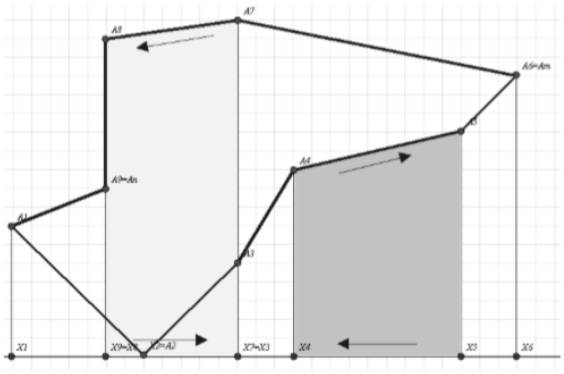


Abbildung 4.5 Darstellung der Trapezzerlegung eines Polygons. Die ermittelte Fläche von A7 nach A8 wird positiv gewertet, während die Fläche von A4 nach A5 negativ ist [2], S. 5.

4.5 Grundlagen aus dem B-Human-Framework

In diesem Unterkapitel werden die zwei entscheidenden Grundvoraussetzungen aus dem B-Human-Framework erklärt, die die Implementierung möglich machen.

4.5.1 Berechnung der sichtbaren Fläche

Die Funktion zur Berechnung der sichtbaren Fläche eines Roboters berechnet ein Polygon aus einer übergebenen Roboterposition, einer Kameramatrix, den Kamerainformationen und den Feldabmessungen. In der Funktion passiert folgendes:
Jeder Punkt des Polygons wird berechnet in dem jeder Endpunkt des Kamerabildes ausgehend von der Roboterposition in das Feld projiziert, was durch die Rückprojektion der Kameragleichung geschieht, und dann über die Höhe skaliert wird. Wenn man das entstehende Polygon einzeichnet, erhält man durch das ständige Wechseln zwischen den Kameras zwei Polygone. Die

durch diese Funktion berechneten Polygone haben immer die gleiche Anordnung von Punkten, siehe Abb. 4.6.

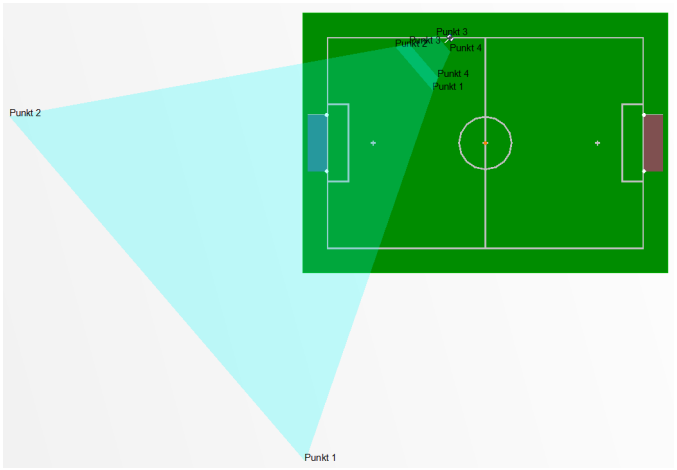


Abbildung 4.6 Das eingezeichnete Sichtfeld eines Roboters.

4.5.2 Ermittlung eines Schnittpunktes

Die Problematik der Schnittpunktbestimmung lässt sich auf eine Formel reduzieren.

$$\begin{pmatrix} a \\ b \end{pmatrix} + \lambda \cdot \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} w \\ x \end{pmatrix} + \mu \cdot \begin{pmatrix} y \\ z \end{pmatrix} \text{ mit } a, b, c, d, w, x, y, z, \lambda, \mu \in \mathbb{R} \quad (4.4)$$

Sei die Lösung, die dabei entsteht wenn man μ und λ berechnet hat, l so gilt:

$$l = \begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} + \lambda \cdot \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} w \\ x \end{pmatrix} + \mu \cdot \begin{pmatrix} y \\ z \end{pmatrix} \text{ mit } m, n \in \mathbb{R}, l \in \mathbb{R}^2 \quad (4.5)$$

In dem Framework löst man diese Gleichungen in dem man überprüft ob die Richtungsvektoren $\begin{pmatrix} c \\ d \end{pmatrix}$ und $\begin{pmatrix} y \\ z \end{pmatrix}$ linear abhängig sind. Ist dies der Fall, gibt es keine eindeutige Lösung oder gar keine. Ist dies nicht der Fall so wird folgendes gerechnet:

$$m = a + c \cdot \frac{b \cdot y - x \cdot y + (-a + w) \cdot z}{(-d) \cdot y + c \cdot z} \quad (4.6)$$

$$n = b + d \cdot \frac{-b \cdot y + x \cdot y + (a - w) \cdot z}{d \cdot y - c \cdot z} \tag{4.7}$$

Zum Vergleich findet sich die hier skizzierte Funktion für die Formel 4.6 und Formel 4.7 im B-Human-Coderelease [11]. Dieser Ansatz ist in den meisten Buildkonfigurationen, das Framework unterscheidet die Buildkonfigurationen nach der betriebenen Optimierung vgl. Kapitel 6.4, deutlich schneller als die reine Lösung des Gleichungssystems aus Formel 4.4 durch die Mathebibliothek Eigen, vgl. [20], bei der man die Gleichung in die folgende Form umwandeln muss:

$$\begin{pmatrix} a - w \\ b - x \end{pmatrix} = \mu \cdot \begin{pmatrix} y \\ z \end{pmatrix} - \lambda \cdot \begin{pmatrix} c \\ d \end{pmatrix} \tag{4.8}$$

Was wiederum für die Eingabe in Eigen zu folgender Formel umgeformt wird:

$$\begin{pmatrix} a - w \\ b - x \end{pmatrix} = \mu \cdot \begin{pmatrix} y \\ z \end{pmatrix} + \lambda \cdot \begin{pmatrix} -c \\ -d \end{pmatrix} \tag{4.9}$$

In der Entwicklung wurden beide Ansätze getestet wobei der Geschwindigkeitsunterschied in der Buildkonfiguration Release nicht so erheblich war. Dabei lässt aber die Präzision der Eigenversion zu wünschen übrig und fordert mehr Genauigkeit bei dem Test ob der Schnittpunkt in beiden Segmenten ist. Daher wird in der finalen Version auf die Funktion aus dem B-Human Framework gesetzt, da diese präzise und schnell funktioniert.

Kapitel 5

Ermittlung der Schnittpunkte

Um zu vermeiden dass zwei Roboter die gleiche Fläche des Feldes abdecken muss man die Schnittpunkte ermitteln, um auf diese Weise Überschneidungen zu detektieren.
Für die Ermittlung der Schnittpunkte wird die Menge aller Liniensegmente in dem aktuellen Weltzustand benötigt. Daher wird die Ermittlung der Schnittpunkte in zwei Aktionen unterschieden:

1. Das Berechnen der Segmente.
2. Das Berechnen der Schnitte.

5.1 Berechnen der Segmente

Die Segmente erhält man durch die Sichtfelder und die Feldgrenzen, wobei in dieser Anwendung die Feldgrenzen die Teppichgrenzen sind. Ein Argument, dass man sich für die Teppichgrenzen als Feldgrenzen ausspricht ist, dass auch in diesem Außenbereich interessante Ereignisse geschehen, die man erkennen könnte, wie zum Beispiel ein gegnerischer Spieler, der nach seiner Strafe in das Spielfeld einläuft oder dort seine Strafe absitzt.
Die Sichtfelder sind wie in Abb. 4.6 gezeigt aufgebaut. Durch diese konsistente Anordnung der Punkte ist die Berechnung der Segmente für ein Sichtfeld einfach. Im Rahmen dieser Implementierung ist ein Segment ein Konstrukt

was durch einen linken und einen rechten Endpunkt definiert ist. Für spätere Operationen besitzt jedes Segment eine ID, die dieses einem Sichtfeld zuordnet und eine ID, die zur Identifikation in einem Sichtfeld dient. Um nun die Segmente des Sichtfeldes zu bestimmen muss man eine Liste von zwei Punkte erstellen, die durch eine Linie verbunden sind und sie dem Konstruktor mit den IDs zu übergeben. Die Einordnung für den linken und rechten Endpunkt erfolgt durch eine geeignete Sortierung, die für die spätere Verwendung des Scanline-Verfahrens entscheidend ist.
Ein Kompromiss in dieser Implementierung ist die Berechnung der Sichtfelder anderer Roboter. Da diese Berechnung eines Sichtfeldes über die Roboterposition und die Kameradaten erfolgt, ist es wichtig dass jeder Roboter diese Daten mit seinen Teampartnern kommuniziert. Eine reine Berechnung auf Basis der bloßen Teampartnerposition trifft eine zu starke Annahme, dass die Teampartner genau so gucken wie man selbst. Dennoch trifft diese Annahme in vielen Fällen erstaunlich gut zu, aber versagt in einigen Fällen extrem. Zum Beispiel, wenn der Stürmer versucht den Sichtbereich des Torwarts zu berechnen, wenn diese gerade in das Feld einlaufen. Daher wird zusätzlich die kommunizierte Kameramatrix benötigt, die durch ihre Rotation und Translation die korrekte Sichtfeldberechnung ermöglicht. Damit besteht das Risiko mit leicht veralteten Sichtbereichen zu rechnen, wobei diese auch von unterschiedlichen Kameras sein können. Konkret bedeutet dies, dass es vorkommen kann, dass der Stürmer mit dem Sichtfeld seiner oberen Kamera rechnet, aber eine maximale Abdeckung zusammen mit dem Sichtfeld der unteren Kamera des Torwarts erreichen möchte. Vergleiche mit Abb. 5.1.
Die Segmente für die Teppichgrenzen werden aus den fest implementierten Teppicheckpunkten aus dem Modul für die Informationen über die Feldausmaße ermittelt in dem die Punkte analog wie bei den Sichtfeldern in Listen gepackt werden.
Während man die Segmente der einzelnen Sichtfelder ermittelt, erhält man auch die offensichtlichen Punkte für das begrenzte Sichtfeld (welches durch die Teppichgrenzen begrenzt wird). Man überprüft für jeden Punkt eines Sichtfelds, ob dieser auf dem Feld ist und speichert, wenn dies der Fall sein sollte, den Punkt in das Polygon für das begrenzte Sichtfeld. Ebenso sind die Eckpunkte des Feldes zu betrachten, da sie Teil des begrenzten Sichtfeldes sind, wenn diese innerhalb des Sichtfeldes sind. So wird jeder Eckpunkt in das Polygon des begrenzten Sichtfeldes aufgenommen, der bei diesem Test

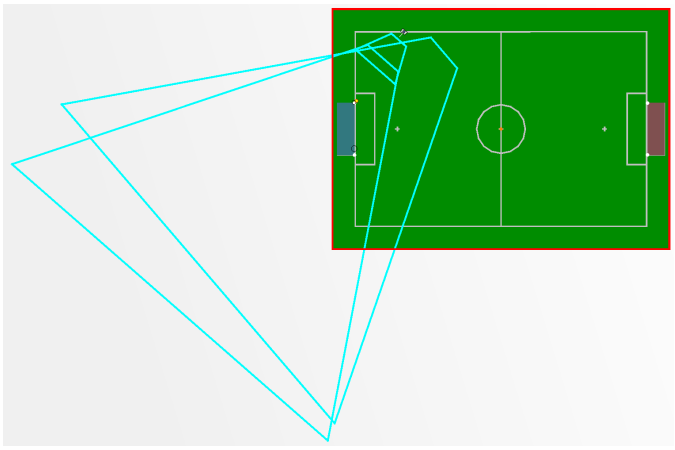


Abbildung 5.1 Die eingezeichneten Segmente. Die Segmente die aus den Sichtfeldern der Roboter entstehen sind blau gefärbt und die Teppichgrenzen sind rot.

ein positives Ergebnis erzielt. Der Test wird im folgenden Abschnitt näher beschrieben.

5.2 Test auf Sichtbarkeit

Der Test auf Sichtbarkeit besteht aus zwei einfachen Linearkombinationen die überprüfen, ob ein Punkt innerhalb eines Trapezes erreicht werden kann. Dieser Test ist vor allem wegen der konsistenten Anordnung der Eckpunkte möglich. Dabei beschreibt man Segmente als Vektoren, wobei die verwendeten Vektoren den selben Ursprungspunkt haben. Der zu überprüfende Punkt wird ebenfalls als Vektor mit dem gleichen Ursprung dargestellt. Seien die Vektoren a und b die Vektoren zu den Segmenten und p der zu prüfende Punkt so lässt man das folgende Gleichungssystem lösen:

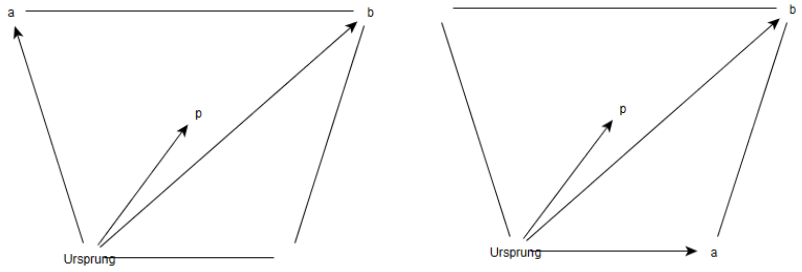
$$a \cdot \lambda + b \cdot \mu = p \text{ mit } \lambda, \mu \in \mathbb{R}, a, b, p \in \mathbb{R}^2 \tag{5.1}$$

Dabei ist für unsere Zwecke eine Lösung nur gültig wenn auch gilt:

1. $0 \leq \lambda, \mu \leq 1$
2. $0 \leq \lambda + \mu \leq 1$

Ohne diese Einschränkung würden auch Punkte die sich hinter dem Sichtfeld

befinden ein positives Ergebnis zurückliefern. Das Lösen erfolgt, wie schon im Unterkapitel der Grundlagen aus dem Framework erwähnt, durch die Mathebibliothek Eigen, wobei es zu Ungenauigkeiten kommen kann und deswegen zusätzlich empirisch ermittelte Schwellwerte genutzt werden. Da aber in einem Trapez mit so einer Gleichung nicht das ganze Trapez abgedeckt wird, muss eine zweite Gleichung in ähnlicher Form analog gelöst werden. Wenn eine Lösung der beiden Gleichungen gültig ist, gibt es ein positives Resultat. Die Idee, die hinter diesen unterschiedlichen Gleichung steckt, ist die Dreieckzerlegung des Trapezes, da man durch ein Gleichungssystem aussagen kann ob ein Punkt in einem Dreieck ist. Somit lässt sich das Problem wie in der Abb. 5.2 beschreiben.



(a) (b)

Abbildung 5.2 Visualisierung der Gleichungssysteme, die durch die Formel 5.1 gelöst werden mit dem Sichtfeld als Trapez.

5.3 Berechnen der Schnitte

Die Berechnung der Schnitte erfolgt mit dem in den Grundlagen vorgestellten Scanline-Verfahren. Wie dort erwähnt basiert die Implementierung auf der Vorlesung [15]. Da in dem Lehrbuch [4] zwar eine schöne allgemeine Problemformulierung vorhanden ist, aber dort Annahmen zur Vereinfachung gemacht werden, die eine Anwendung in dieser Domäne unmöglich machen. Um also die interessanten Schnittpunkte der Sichtfelder miteinander und mit den Teppichgrenzen zu ermitteln wird das Verfahren, was in diesem Unterkapitel beschrieben wird, verwendet.

Sei die Menge der Segmente aus dem vorletzten Unterkapitel der Segmentbeschaffung gegeben so werden drei Rot-Schwarz-Bäume benutzt. Dies sind die Eventqueue Q (ein Baum für die abzulaufenden Punkte), die Statusqueue T (ein Baum für die zu betrachtenden Segmente) und die Schnittpunkte I (ein Baum für die Schnittmarken). Eine Schnittmarke ist ein Konstrukt, das aus einem Schnittpunkt besteht und diesem die zwei Segmente zuordnet, die diesen entstehen lassen. Die Eventqueue wird mit den Anfangs- und Endpunkten aller Segmente gefüllt, die in diesem Baum nach der Größe des x-Wertes sortiert werden. Durch diese Sortierung arbeitet sich dieser Algorithmus von links nach rechts über die Spielfeldebene.
Solange die Eventqueue Q nicht leer ist wiederhole:
Entferne den nach der Sortierung niedrigsten Wert aus Q, der jetzt zu unserem betrachteten Punkt p wird. Aus dem Punkt p werden die zwei Segmentmengen zur Schnittpunktuntersuchung hergeleitet. Diese sind die Mengen Up und Lp, nach Literatur gibt es auch die Menge Cp die nur zur Veranschaulichung hier aufgeführt wird. Da diese Bezeichnungen aus der Literatur [15] übernommen wurden, muss hier ein bisschen umgedacht werden, da wir hier anders mit der Scanline über die Ebene fahren.

Segmentmenge	Bedeutung nach Literatur	Bedeutung in dieser Implementierung
Up	Die Menge der Segmente die p als oberen Endpunkt haben.	Die Menge der Segmente die p als linken Endpunkt haben.
Lp	Die Menge der Segmente die p als unteren Endpunkt haben.	Die Menge der Segmente die p als rechten Endpunkt haben.
Cp	Die Menge der Segmente die p in ihrem Bereich der y-Werte haben.	Die Menge der Segmente die p in ihrem Bereich der x-Werte haben. **Wird nicht verwendet!**

Tabelle 5.1 Übersicht der Mengen für die Schnittpunktfindung.

Up wird ermittelt in dem die übergebene Segmentmenge durchlaufen wird und alle Segmente mit entsprechender Eigenschaft übernommen werden. Die Menge Lp wird ähnlich über die Statusqueue T ermittelt, da jedes Segment was noch aktiv ist, also auf der Scanline liegt, nur diesen Punkt p als rechten Endpunkt haben kann. Alle anderen Segmente sind entweder nicht mehr auf der Scanline oder werden erst noch in diese aufgenommen. Um zu überprüfen ob dieser bestimmte Punkt schon ein gemeinsamer Schnittpunkt

von Segmenten ist, bildet man die Vereinigung von Up und Lp. Ist die Vereinigung größer als 1, dann schneiden sich in diesem Punkt Segmente. Um all diese Schnittpunkte in Schnittmarken zusammenzufassen wird in der Vereinigung für jedes Segment mit jedem Anderen eine Schnittmarke erstellt und in die Schnittpunkte I eingefügt. Dabei wird in dieser Schnittmarke das erste Segment immer das mit der niedrigsten ID des dazugehörigen Polygons sein. Diese Konstruktion wird später bei der Flächenberechnung eine Rolle spielen.
Im Anschluss wird jedes Element aus der Menge Up auf Schnitte mit den Segmenten in der Statusqueue T überprüft. Dabei gilt ein Schnittpunkt als gefunden wenn genau eine Lösung in der Geradengleichung existiert und der Schnittpunkt innerhalb der beiden Segmente liegt, die man auf den Schnitt getestet hat. Dieser Schnittpunkt wird zu einer Schnittmarke zusammengefasst und in die Schnittpunkte I eingefügt. Dabei gilt auch hier, dass in der Schnittmarke das erste Segment immer das mit der kleinsten Polygon-ID ist.
Der Test ob ein Punkt x auf einem Segment liegt, berechnet ob ein Vektor von dem linken Endpunkt des Segments zu dem Punkt x, die selbe Richtung hat, wie ein Vektor vom linken Endpunkt zum rechten Endpunkt des Segments und dabei kleiner ist als der Segmentvektor. Dabei wurde für diesen Gleichheitstest empirisch ein Schwellwert für den Unterschied der Richtung ermittelt, damit kleinere Ungenauigkeiten nicht korrekte Schnittpunkte verschwinden lassen. Während der Entwicklung wurde analog zur Schnittpunktberechnung in dem Kapitel der Grundlagen eine Version dieses Tests auf Basis der Mathebibliothek Eigen entwickelt, aber ebenso wie beim Schnitttest hat sich dieser selbst mit Hilfe von Schwellwerten als unzureichend präzise und zu aufwändig herausgestellt.
Nach diesem Test auf Schnittpunkte wird die Statusqueue T auf den aktuellen Punkt p gesetzt und im Anschluss der ganze verbleibende Baum aktualisiert in dem alle Elemente aus Lp aus T entfernt werden und alle Elemente aus Up in T eingefügt werden.
Wenn die Eventqueue Q leer ist, werden die Schnittpunkte I in ihrer Rot-Schwarz-Baum-Form zurückgeliefert. In der Simulation sieht das Resultat dann wie in der Abb. 5.3 aus.

Ein Unterschied zur Vorlesung [15] ist, dass in dieser Implementierung wesentlich mehr auf Schnitte getestet wird. In einer vorherigen Version des

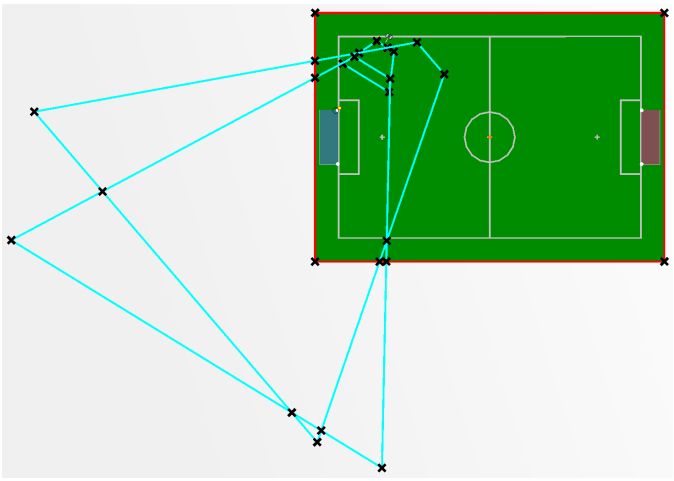


Abbildung 5.3 Die eingezeichnete Menge aller gefundenen Schnittpunkte.

Scanlines, die komplett auf die Vorlesung aufsetzte, wurde nur ein Bruchteil der Punkte gefunden. Die besonders interessanten Punkte, die Schnittmarken deren beteiligte Segmente unterschiedliche Polygon-IDs haben, wurden in absoluten Ausnahmefällen gefunden. Eine Vermutung ist, dass dieser Algorithmus eher darauf setzt, dass je nach Scanrichtung höchstens ein vertikales oder horizontales Segment existiert und bearbeitet werden kann, aber wir wegen der Feldgrenzen mindestens zwei von jeder Sorte haben. Daher erschien es sinnvoll jedes Element, was neu auf die Scanlinie kommt, mit allen vorhandenen zu verrechnen. Ebenso wird hier nicht primär auf die Identifikation der Schnitte über die Behandlung der vereinigten Mengen gesetzt, sondern durch die Entdeckung der Schnitte durch Überprüfung, weshalb hier nur die Vereinigung von Up und Lp betrachtet wird. Deshalb wird auch keine strenge Überprüfung für neue Events vorgenommen. Nach Vorlesung müsste bei einem gefundenen neuen Event überprüft werden ob dieses sich hinter oder auf der Scanlinie befindet, was hier nicht benötigt wird, da gefundene Schnittpunkte nicht als Events behandelt werden, weshalb auch nicht die Menge Cp benötigt wird. Durch diese Abwandlungen des ursprünglichen Algorithmus wird in manchen Gebieten der Aufwand verringert und in einigen erhöht. Durch die Schnittüberprüfung wird zum Beispiel der Aufwand massiv erhöht, da in der Vorlesung nur konstant viele Überprüfungen pro Schritt fällig werden, aber hier besteht die Möglichkeit dass quadratisch viele Überprüfungen benötigt werden. Jedoch werden kleinere Aufwände, wie die Erstellung der Menge Cp und die Überprüfung auf Neuheit des Events gespart. Kritisch betrachtet kann man nun behaupten,

dass die Schnittüberprüfung an der Grenze der Echtzeitberechenbarkeit liegt, da der Worstcase in $O(n^2)$ liegt. Dieser kann erzeugt werden wenn man nur gleichlange horizontale Segmente überprüft. Echtzeitberechenbarkeit ist im Kontext von B-Human die Abarbeitung aller 60 Bilder in einer Sekunde, was für die Gesamtleistung der Roboter fundamental ist. Das Argument warum dieser Ansatz doch im Vergleich zu der Brute-Force-Methode besser geeignet ist, ist die Tatsache das dieser Fall nie eintreten wird, da durch die Feldgrenzen nie alle Segmente in der Statusqueue T sein können. Dieser Vorteil hat sich auch bei den Laufzeiten der Algorithmen gezeigt.

Kapitel 6

Berechnung der Fläche

Dieses Kapitel erklärt wie nach der Berechnung der Schnittpunkte die kooperativ sichtbare Fläche berechnet wird.

6.1 Zuordnung der Punkte

Um das durch die Schnittpunkte gewonnene Wissen zu nutzen, bedarf es der Interpretation von eben jenen. Dazu ist das Konstrukt der Schnittmarke mit der Segmentzuordnung hilfreich und die Unterscheidung zwischen erstem und zweitem beteiligtem Segment. Um diese ganzen Schnittpunkte korrekt, einzuordnen wird eine Schnittmappe erstellt. Eine Schnittmappe ist ein zweidimensionales Array, welches Listen von Schnittmarken enthält. Dabei hat die Mappe 5×5 Zellen, da 5 die Anzahl der maximal aktiven Roboter für das Team ist und jeder solcher Roboter nur ein (aktuell betrachtetes) Sichtfeld hat.
Die Schnittmarken werden als ein Rot-Schwarz-Baum zurückgeliefert. Dieser Baum kann als sortierte Liste angesehen werden und wird hier auch als eine solche behandelt. Aus dieser Liste der Schnittmarken wird immer das kleinste Element herausgenommen und eine Unterscheidung getroffen in dem die IDs der beteiligten Segmentpolygone betrachtet werden. Sollten die Polygon-IDs der beteiligten Segmente gleich sein, bedeutet dies, dass man einen Eckpunkt des Sichtfeldes hat und damit kein Wissen erlangt wurde, da der Punkt durch die Sichtfeldberechnung bereits vollkommen bekannt ist, weshalb dann direkt der nächste Punkt betrachtet wird.

Bei den anderen beiden Unterscheidungen ist es wichtig zu wissen, dass die Sichtfelder der Roboter Polygon-IDs von 1 bis 5 haben, entsprechend der jeweiligen Spielernummer, während die IDs der Feldgrenzen 10 sind, da anzunehmen ist, dass in näherer Zukunft keine 10 Roboter in einem Team spielen.
Sollte dieser Schnittpunkt als erstes schneidendes Segment ein Sichtfeldsegment haben und als zweites Segment eine Feldgrenze so hat man hier einen Punkt, der zum begrenzten Sichtfeld gehört. Deswegen wird dieser Punkt in das Polygon des begrenzten Feldes unter der Polygon-ID des ersten Segments gespeichert. So werden alle begrenzten Sichtfelder ermittelt in dem dieser Fall behandelt wird.
Der dritte Fall trifft zu wenn zwei nicht Feldgrenzen sich schneiden und dieser Schnittpunkt sich auf dem Feld befindet. Diese Schnittmarke wird dann in die Schnittmappe gespeichert wobei die ID des ersten Segments zur Identifikation der Zeile und die ID des zweiten Segments zur Identifikation der Spalte dient. Allein durch diesen Fall wird die Schnittmappe gefüllt.
Wurden alle diese Schnittpunkte bearbeitet, so hat man alle begrenzten Sichtfelder vollständig berechnet und auch die Schnittmappe gefüllt. Die begrenzten Sichtfelder sehen dann wie in der Abb. 6.1 aus.
Nun betrachtet man die Schnittmappe um die Schnittflächen zu bestimmen.

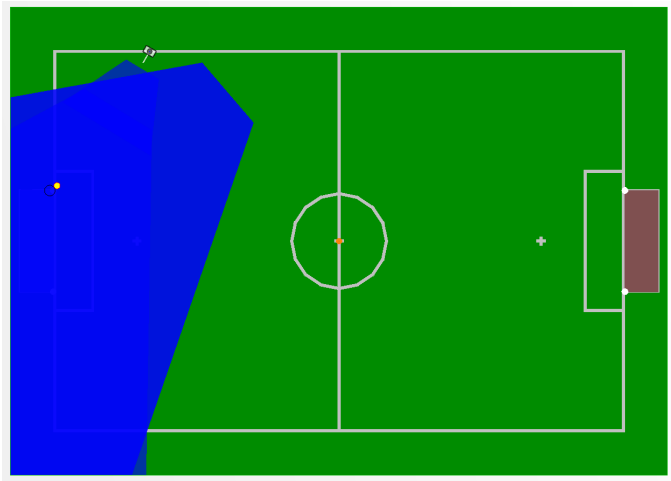


Abbildung 6.1 Die begrenzten Sichtfelder von zwei Robotern.

Wir iterieren über die Schnittmappe, wobei nach der Ordnung nur maximal 10 Einträge relevant sein können. Wie in der Tabelle 6.1 dargestellt.

Während man über die Schnittmappe iteriert wird überprüft ob der jetzige

Polygon-IDs	1	2	3	4	5
1	Irrelevanter Schnitt	Schnitt (1,2)	Schnitt (1,3)	Schnitt (1,4)	Schnitt (1,5)
2	Behandelt in (1,2)	Irrelevanter Schnitt	Schnitt (2,3)	Schnitt (2,4)	Schnitt (2,5)
3	Behandelt in (1,3)	Behandelt in (2,3)	Irrelevanter Schnitt	Schnitt (3,4)	Schnitt (3,5)
4	Behandelt in (1,4)	Behandelt in (2,4)	Behandelt in (3,4)	Irrelevanter Schnitt	Schnitt (4,5)
5	Behandelt in (1,5)	Behandelt in (2,5)	Behandelt in (3,5)	Behandelt in (4,5)	Irrelevanter Schnitt

Tabelle 6.1 Aufbau einer Schnittmappe zur Flächenbildung.

Eintrag leer ist. Ist dies der Fall so gab es keinen Schnitt zwischen den Sichtfeldern und der Eintrag ist nicht von Interesse und wird übersprungen. Sollte der Eintrag nicht leer sein, so wird eine Schnittfläche durch ein Polygon aufgebaut. Dafür greift man auf die begrenzten Sichtfelder der Roboter zu und packt alle Punkte dieser Polygone in das Schnittpolygon die jeweils in dem Sichtfeld des Anderen sind. Anschließend wird das resultierende Polygon um die Punkte der Schnittmarken an der jetzigen Stelle der Schnittmappe erweitert. Ist dies geschehen so kann man das resultierende Polygon in die Liste der Schnittflächen speichern und den nächsten Eintrag der Schnittmappe betrachten. Die Schnittflächen sehen dann wie in der Abb. 6.2 aus. Hat man alle Stellen der Schnittmappe betrachtet so kann man nun mit den begrenzten Sichtfelder und den Schnittflächen die Gesamtabdeckung berechnen.

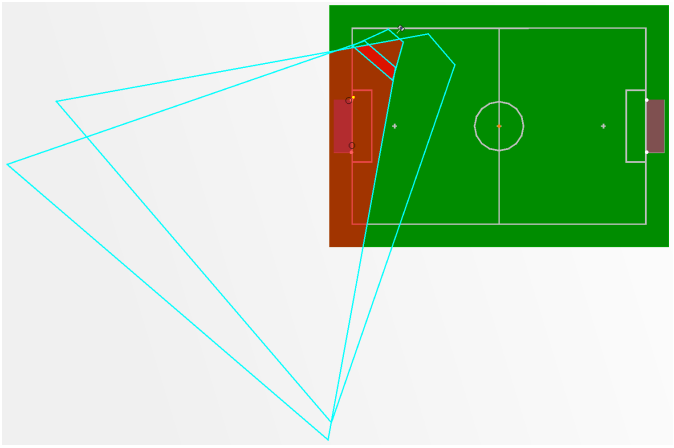


Abbildung 6.2 Die erzeugte Schnittfläche zwischen zwei Robotern.

6.2 Berechnung der Abdeckung

Die Berechnung der Abdeckung erfolgt nun aus den ermittelten Schnittpolygonen und begrenzte Sichtfeldern. Dabei werden erst die Flächen der begrenzten Sichtfelder aufsummiert, wobei die Polygone noch sortiert werden müssen, damit die Gaußsche Trapezformel ein korrektes Ergebnis liefert.
Die Sortierung soll dabei erreichen, dass sämtliche Punkte so angeordnet sind, dass der Rand des Polygons in eine Richtung umlaufen wird. Bei der Sortierung wird für ein Polygon der arithmetische Mittelpunkt berechnet. Dann werden die Punkte nach dem Winkel zu diesem Mittelpunkt angeordnet. Dadurch wird eine für die Gaußsche Trapezformel geforderte Ordnung erreicht und es kann somit ein Wert für die Fläche berechnet werden.
Für die Schnittflächen muss man berücksichtigen, dass sich diese auch gegenseitig schneiden können. Da man aber dies nicht richtig, beziehungsweise nur mit enormen Aufwand prüfen kann, muss man dies möglichst präzise schätzen.
Um mit etwas mehr Präzision zu schätzen, wird aus der Menge der Schnittpolygone das erste Element herausgesucht und alle anderen auf die Nähe zu diesem überprüft. Dabei bedeutet Nähe, dass die arithmetischen Mittelpunkte dieser Polygone sich in einer Distanz von 30 cm oder weniger zueinander befinden. Von allen Schnittpolygonen, die dieses Kriterium erfüllen, werden die Punkte sortiert und die Fläche berechnet. Am Ende wird die größte Schnittfläche von der bisherigen Summe der Flächenabdeckung abgezogen. Während des Verfahrens werden die weiter entfernten Schnittpolygone in eine separate

Liste gepackt und am Ende jeder Berechnung der größten Schnittfläche wird die Menge der Schnittpolygone durch diese Menge der weiter entfernten Polygone ersetzt. Dieses Verfahren wird solange wiederholt bis es keine Schnittpolygone mehr gibt. Als Resultat hat man dann die im Moment abgedeckte Fläche.
Ein Phänomen was durch das Kommunizieren der Kameradaten entsteht ist, das bei dem hochfrequentem Wechsel der Kameras des aktuell betrachteten Roboters die Abdeckung konsistent mit den gerade vorhandenen Sichtfeldern der anderen Roboter berechnet wird und somit zwei Werte entstehen. So kann es sein dass bei dieser Vorgehensweise, wie in der Abbildung Abb. 6.3 dargestellt, der Unterschied zwischen den beiden Werten gering ist, wenn alle anderen Roboter die obere Kamera verwenden und einen Großteil des eigenen Sichtbereichs abdecken.

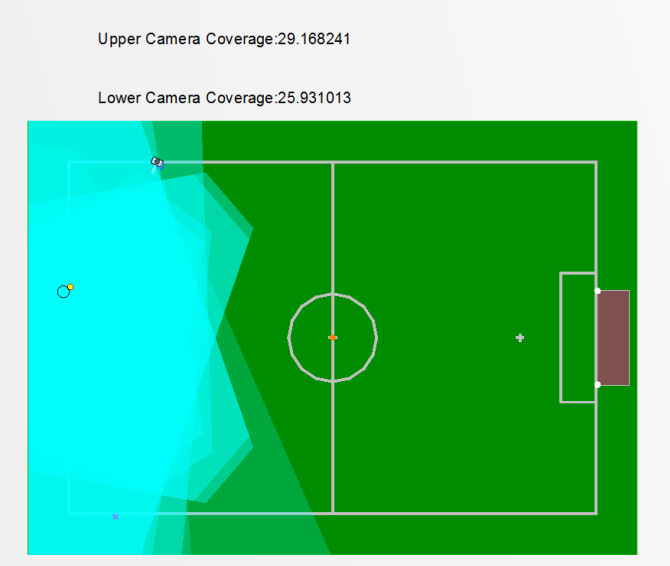


Abbildung 6.3 Visualisierung des Ergebnisses für die Flächenabdeckung bei 5 Robotern. Die Werte in der Zeichnung über dem Feld sind in m^2.

6.3 Die Maximierung

Im Rahmen der Kopfsteuerung wird die Maximierung durch einen Zustandsautomaten erreicht. Dieser Zustandsautomat ist sehr klein und besitzt nur zwei Zustände und wird nur benötigt um nicht permanent den Aufwand der Maximierung

zu haben. Dieser Zustandsautomat ermöglicht es auch nicht echtzeitfähige Algorithmen auf dem Roboter mit akzeptablen Nebeneffekten laufen zu lassen und wurde vorher stark bei der nicht derart optimierten Version dieser Kopfsteuerung benötigt. Der erste Zustand ist der Zustand in dem keine Berechnung erfolgt ist und dieser wird nur ausgeführt wenn eine gewisse Zeit lang auch keine Berechnung statt gefunden hat. In diesem Fall werden die Sichtfelder der Roboter berechnet. Dabei wird die Berechnung nur fortgeführt, wenn alle Sichtfelder von der oberen Kamera sind. Ist dieser Fall ebenfalls gegeben wird der arithmetische Mittelpunkt des aktuellen Sichtfeldes vorläufig als Punkt zur Maximierung gesetzt. Anschließend befindet man sich in dem zweiten Zustand in dem man das eigene Sichtfeld um verschiedene Winkel dreht, um so eine maximale Abdeckung zu erreichen. Dabei werden die Winkel relativ gering gewählt, damit eine möglichst präzise Maximierung erreicht wird. Ein verändertes Sichtfeld wird mit den Sichtfeldern der anderen Roboter zur Flächenberechnung übergeben. Wenn die Fläche größer ist als die bisher erhaltene Fläche, wird der Mittelpunkt des neuen Sichtfeldes als Punkt der Maximierung übernommen. Wenn man hier größere Winkel nimmt wird nicht unbedingt die beste Abdeckung erreicht, sondern nur eine sehr grobe Annäherung. Durch das wiederholte Nutzen der Berechnung werden aber auch größere Winkel zur Maximierung erzeugt.

6.4 Aufwand der einzelnen Aktionen

Wie in der Einleitung erwähnt gelten geometrische Berechnungen im Bereich der Robotik als sehr rechenintensiv. Die grundlegenden Funktionen zur Maximierung der Fläche habe ich mit einer Funktion aus dem B-Human Framework überwacht, um ihre Rechenzeit zu ermitteln. Dabei kamen in der Simulation mit 5 Robotern, auf einem i7-Desktoprechner, bei den Buildkonfigurationen Release und Develop aus Sicht eines Roboters folgende Werte in Millisekunden heraus:

Funktionen	**Develop (ms)**			**Release (ms)**		
	MIN	MAX	AVG	MIN	MAX	AVG
Berechnung der Sichtfelder	0,18	0,46	0,36	0,003	0,01	0,005
Berechnung der Abdeckung(Fläche)	0,23	10,51	1	0,005	0,011	0,005
Berechnung der Segmente	2,5	10,43	4,17	0,02	0,04	0,02
Berechnung der Schnittpunkte	5,03	26,33	12,04	0,09	0,3	0,2
Berechnung der Schnittflächen	11,37	40,76	15,93	0,08	0,23	0,16
Berechnung einer Feldabdeckung	21,57	50,89	33,54	0,19	0,56	0,37
Berechnung eines Maximierungsschrittes	0	45,08	1,63	0	0,51	0,02

Tabelle 6.2 Laufzeiten der unterschiedlichen Funktionen mit 5 simulierten Robotern auf einem i7-Desktoprechner.

Auf einem echten Roboter (Bernadette):

Funktionen	**Develop (ms)**			**Release (ms)**		
	MIN	MAX	AVG	MIN	MAX	AVG
Berechnung der Sichtfelder	0,01	0,05	0,01	0,008	0,04	0,01
Berechnung der Abdeckung(Fläche)	0,01	0,05	0,02	0,01	0,05	0,02
Berechnung der Segmente	0,04	0,12	0,05	0,04	0,1	0,05
Berechnung der Schnittpunkte	0,15	0,4	0,23	0,15	0,5	0,23
Berechnung der Schnittflächen	0,04	0,15	0,07	0,04	0,19	0,07
Berechnung einer Feldabdeckung	0,26	0,66	0,41	0,26	0,63	0,37
Berechnung eines Maximierungsschrittes	0	0,62	0,02	0	0,63	0,02

Tabelle 6.3 Laufzeiten der verschiedenen Funktionen auf einem echten Roboter.

Die Laufzeitwerte die aus Sicht eines Roboters mit einem weiteren aktiven Teammitglied aus den Logs extrahiert wurden (45781 Werte):

Funktionen	**Release (ms)**		
	MIN	MAX	AVG
Berechnung der Sichtfelder	0	76	0,46
Berechnung der Abdeckung(Fläche)	0	159	1,32
Berechnung der Segmente	0	196	2,66
Berechnung der Schnittpunkte	0	762	13,39
Berechnung der Schnittflächen	0	378	4,63
Berechnung einer Feldabdeckung	0	1233	17,12
Berechnung eines Maximierungsschrittes	0	1178	6,2

Tabelle 6.4 Laufzeiten aus einem Log extrahiert bei zwei aktiven Robotern.

Verhältnis der Laufzeiten für die Berechnung der Feldabdeckung mit unterschiedlich vielen Robotern auf dem Laptop (i7):

Anzahl der Roboter	**Release (ms)**		
	MIN	MAX	AVG
1	0,045	0,472	0,08515
2	0,166	0,892	0,21688
3	0,266	1,201	0,33184
4	0,365	2,143	0,51944
5	0,455	3,346	0,69576

Tabelle 6.5 Laufzeiten für die Berechnung der Feldabdeckung bei unterschiedlich vielen Robotern auf einem i7-Laptop.

Zu beachten ist dass diese Werte nur eine Momentaufnahme sind und nicht 100%-ig zueinander passen müssen, da sie sich permanent verändern. So sind manche Operationen relativ präzise überwacht worden, wie die Berechnung der Feldabdeckung, da diese absolut kontinuierlich ausgeführt werden. Während bei anderen Operationen diese Präzision nicht erreicht werden konnte, wie zum Beispiel bei der Berechnung eines Maximierungsschrittes. Dadurch können Phänomene auftreten, wie die im Vergleich höheren Werte bei der Berechnung der Feldabdeckung zu der Berechnung eines Maximierungsschrittes, die die erstere Berechnung auf jeden Fall ebenfalls in sich aufruft. Somit muss die Aufsummierung der Spalten mit dem Ergebnis der Berechnung der Feldabdeckung auch nicht unbedingt übereinstimmen. Dennoch geben die Messungen eine ganz gute Übersicht über die Ausführungszeiten.
Um die Unterschiede in den Buildkonfigurationen zu verdeutlichen befindet

sich in der Tabelle 6.6 eine Übersicht. Dadurch erklärt sich auch warum der Unterschied zwischen Develop und Release auf dem *NAO* sehr gering und in der Simulation auf dem Desktoprechner sehr groß ist, obwohl der Unterschied in der Rechenkapazität eines i7-Prozessors zu einem Intel ATOM enorm ist.

	Besitzt Asserts (NDEBUG)	Debug-Symbole (Compiler-Flags)	Optimierungen (Compiler-Flags)
Release			
Nao	×	×	✓
SimulatedNao	×	×	✓
Develop			
Nao	✓	×	✓
SimulatedNao	✓	✓	×

Tabelle 6.6 Übersicht der Unterschiede zwischen den Buildkonfigurationen Develop und Release.
Nao ist die B-Human-Anwendung für den *NAO*.
SimulatedNao ist eine shared library die den B-Human-Code für den Simulator enthält.
Diese Tabelle ist von [11] ins Deutsche übersetzt worden.

Kapitel 7

Die Kopfsteuerung

Obwohl diese Arbeit die Maximierung der kooperativ sichtbaren Fläche behandelt, ist dies nicht immer der ideale Ansatz für eine Kopfsteuerung. Wie schon in der Einleitung erwähnt, soll diese Arbeit eine Kopfsteuerung für das Team B-Human liefern, aber für diese Kopfsteuerung wird die Maximierung dennoch ein wichtiges Feature sein.
Um sich aber vor Augen zu führen warum die Maximierung nicht permanent ausgeführt werden soll muss man sich die Roboter in einem Spiel vorstellen. Die Roboter bewegen zwar ihre Köpfe um sich zu lokalisieren und neue Ereignisse wahrzunehmen dennoch muss man auch Kopfbewegungen für bestimmte Aktionen parat haben. Man stelle sich vor ein Mensch würde den ganzen Tag in einer nicht ruckartigen Bewegung seinen Kopf immer von links nach rechts und wieder zurück drehen. Besagter Mensch würde zwar die meisten Ereignisse in seiner Umgebung mit bekommen jedoch hätte er Schwierigkeiten Aktionen wie das Öffnen einer Tür oder das Verfassen eines Textes durchzuführen. Dabei sprechen wir nur von Schwierigkeiten und nicht von einer Unmöglichkeit, da der Mensch den Tastsinn besitzt und darüber Informationen erhält und dadurch die fehlenden visuellen Informationen ausgleicht. Bei den Robotern verhält es sich ähnlich, nur dass die Sensorik nicht den kompletten Tastsinn ersetzen kann. Die Aufgaben, die unsere Roboter im Bereich der SPL lösen müssen, haben allerdings wenig mit dem Tastsinn zu tun. Dass aber das erklärte Phänomen des Datenverlusts durch gezielte Kopfsteuerung auch hier zutreffen kann zeigt folgendes Beispiel: Man nehme an, ein Roboter des Teams habe sich im Verlauf des Spiels

den Ball erkämpft und ist jetzt in der Rolle des Stürmers. Der Stürmer hat das Ziel den Ball nach vorne in die gegnerische Hälfte zu bringen und ein Tor zu schießen. Dabei sollte er auch Teampartner beachten die in der gegnerischen Hälfte freistehen und sich somit für einen Pass anbieten. Im Falle der Kopfsteuerung, die nur die kooperativ sichtbare Fläche maximiert, kann es sein, dass alle anderen Roboter zu $\frac{3}{4}$ das Feld abdecken, da drei Roboter in der eigenen Hälfte stehen und diese nahezu optimal abdecken während der verbleibende Roboter in der rechten Spielfeldhälfte der Gegner steht und diese abdeckt. Sei unser Stürmer nun exakt in der Mitte des Feldes und sollte nur den Ball nach vorne bringen, könnte nach dem Prinzip der Maximierung der kooperativ sichtbaren Fläche von ihm verlangt werden, dass er statt den Ball im Blick zu halten lieber die linke Spielfeldhälfte der Gegner abdeckt. Somit würden ihm die aktuellen Informationen über den Verbleib des Balles entgehen, beziehungsweise werden die Informationen nur verspätet über die Teampartner bezogen, die den Ball gerade im Sichtfeld haben. So sinkt die Wahrscheinlichkeit, dass man über längere Zeit im Ballbesitz bleibt oder gar ein Tor schießt. Dies ist nur eines von vielen Szenarien in denen das Bestreben der Maximierung der kooperativ sichtbaren Fläche eher hinderlich als förderlich ist. Um dieses Problem zu umgehen wird die Maximierung im Rahmen einer prioritätsgesteurten Kopfsteuerung benutzt.

7.1 Aufbau der Kopfsteuerung

Die in dem Rahmen dieser Arbeit entstandene Kopfsteuerung verfolgt den Ansatz bestimmte Ziele hin und wieder beobachten zu wollen. Man kann sich vorstellen dass es für Fußball spielende Roboter essentiell ist den Ball zu sehen, aber auch andere Ziele wie die Beobachtung der Gegner und der Teampartner können ausschlaggebend sein für den Erfolg. Dennoch können diese Ziele unterschiedlich wichtig sein weshalb eine Priorisierung erdacht werden muss. Die Zielklassen dieser Steuerung, in die Objekte einsortiert werden, sind: Ball, Flächenabdeckung, Gegner, Teampartner, Hindernisse und Feldpositionen. Die Begründungen warum der Ball, die Gegner und die Teampartner in dieser Liste stehen sind offensichtlich und wurden auch zum Teil skizziert. Die Flächenabdeckung, die in dieser Arbeit untersucht wird,

ist auch in dieser Liste vorhanden, während die Hindernisse aus dem Kontext B-Human gewachsen sind, da alle Objekte die sich auf dem Feld befinden, aber nicht klassifiziert werden können, als Hindernis klassifiziert werden. Die Feldpositionen sind ein Überbleibsel aus der vorherigen Kopfsteuerung. Diese Feldpositionen sind feste Punkte, die sich aus dem nach dem Regelwerk [12] definierten Feld ableiten und für die Lokalisierung des Roboters entscheidend sein können. Aus diesen Gründen wurden diese übernommen aber deren Behandlung abgewandelt. Diese Zielklassen unterscheiden sich in der Zielbeschaffung und den Prioritätsbereich, wobei die Prioritäten nach unterschiedlichen Kriterien erhöht oder verringert werden. Hier eine Übersicht:

Zielklasse	**Quelle**	**Prioritätsbereich**	**Kriterien**
Ball	LibBall	1 - 100	Sichtbarkeit des Balles im Team.
Flächenabdeckung	LibMonitor	15	-
Feldposition	Config-Datei	1 - 2	Ziel länger als 100 Sekunden nicht gesehen.
Gegner	ObstacleModel	-1 - 15	Distanz zum Ball ist kleiner als 1,5 Meter.
Teampartner	ObstacleModel	1 - 9	Distanz zum Ball und Sichtbarkeit.
Hindernis	ObstacleModel	0 - 6	Distanz zum Ball.

Tabelle 7.1 Übersicht aller Zielklassen.

Um diese ganzen Ziele zu verwalten und zu speichern, gibt es zwei Datenstrukturen. Eine Prioritätswarteschlange dient dazu die ganzen Ziele für den Roboter nach Priorität aufzulisten und wird im Verlauf der Steuerung verkleinert. Ziel ist es also diese Struktur für den Roboter sinnvoll zu befüllen. Um dies zu bewerkstelligen wird zusätzlich ein Array von Ziellisten verwaltet. Dieses Array ist so groß wie die Anzahl der Zielklassen, somit hat es 6 Einträge. Jeder Eintrag repräsentiert eine Zielklasse und verwaltet in der Liste all seine Vorkommen. Dabei ist die Anzahl der Vorkommen nicht immer gleich. So geht man in dieser Implementierung davon aus, dass immer nur ein Ball vorhanden ist und analog wird auch die Flächenabdeckung gehandhabt. Für jede dieser Klassen gibt es eine Beschaffungsfunktion, die den Arrayeintrag initial befüllt und eine Updatefunktion, die die beschafften

Daten auf den aktuellen Stand hält. Dabei sind die Berechnungsfunktionen für den Ball und die Flächenabdeckung auch deren Updatefunktionen. Die Beschaffungsfunktionen werden in den anderen Klassen nur noch dann aufgerufen wenn sich die Anzahl der erhalten Daten ändert.
Zur Aufnahme in die Prioritätswarteschlange müssen zwei Kriterien erfüllt werden. Das Erste ist, dass für die Rolle des Roboters tatsächlich die Zielklasse in seiner Zielspezifikation liegt. Die Zielspezifikation ist eine Liste von Indizes, um auf die Inhalte des Arrays zuzugreifen. Diese Zielspezifikationen sind selber in einem Array gespeichert und der Index für die entsprechende Spezifikation wird für einen Roboter über eine Funktion ermittelt, die jeder Rolle einen Wert zuweist. Das zweite Kriterium ist aus der alten Kopfsteuerung gewachsen und berücksichtigt die Ansteuerbarkeit des Zieles. Dies wird über die Position des Roboters und der Position des Ziels ermittelt und somit eine Grenze in Bezug zur gerade ausgeführten Aktion gewählt. Dies Kriterium ist insofern wichtig, da Ziele die hinter dem Roboter liegen nicht korrekt angesteuert werden können und beim Ansteuern nicht terminieren. Das bedeutet, dass der Roboter seinen Kopf nicht mehr zurücknimmt. Wenn dies geschieht kann es zu bösen Nebeneffekten kommen, wie zum Beispiel Roboter die das Feld verlassen oder sich gegenseitig umrennen.

7.2 Das Ansteuern

Das eigentliche Ansteuern erfolgt über einen Zustandsautomaten. Dabei kann man sich für diese Steuerung aussuchen ob man bei der Ansteuerung der Ziele berücksichtigen will, ob der Ball immer noch im Blickfeld bleiben soll. Der Zustandsautomat überprüft ob die Prioritätswarteschlange noch Ziele für die Steuerung bereithält. Das Ziel mit der höchsten Priorität wird als Position zurückgeliefert, dabei ist diese Position die roboterrelative Objektposition. Wenn aber die Option genutzt wurde, dass nur Ziele angesteuert werden, die den Ball im Sichtfeld behalten, so werden so lange die höchst gewerteten Ziele verworfen bis ein Ziel gefunden wurde, was auf diese Weise erreichbar ist. Dieses Ziel liefert eine roboterrelative Position zurück, die mit dem Ball im Sichtfeld erreichbar ist die während der Zielbeschaffung berechnet wurde, was mit einer Funktion geschieht, die aus der alten Kopfsteuerung übernommen wurde. Wurde das zu überliefernde Ziel ermittelt, so wird das

Array zur Zielverwaltung aktualisiert indem die Priorität um 1 verringert wird. Wurde ein Ziel zurückgeliefert geht der Zustandsautomat dazu über das Ziel anzusteuern. Die Ansteuerung terminiert wenn das Ziel im Sichtfeld ist oder der Winkel der Kopfdrehung mit dem Winkel zum Ziel übereinstimmt. Wenn kein Ziel mehr vorhanden ist guckt der *NAO* erstmal nach links und rechts für zwei Sekunden. Wenn es wieder Ziele gibt wird die maximale Abdeckung für vier Sekunden angeguckt. Danach wird wieder zwei Sekunden nach links und rechts geguckt und anschließend die ganze Prozedur wiederholt. Zum besseren Verständnis kann man die Abb. 7.1 zurate ziehen, die den ganzen Ablauf als Flussdiagramm beschreibt. Dabei wird die Eigenschaft ob der Durchgang abgeschlossen ist intern als boolesche Variable modelliert.

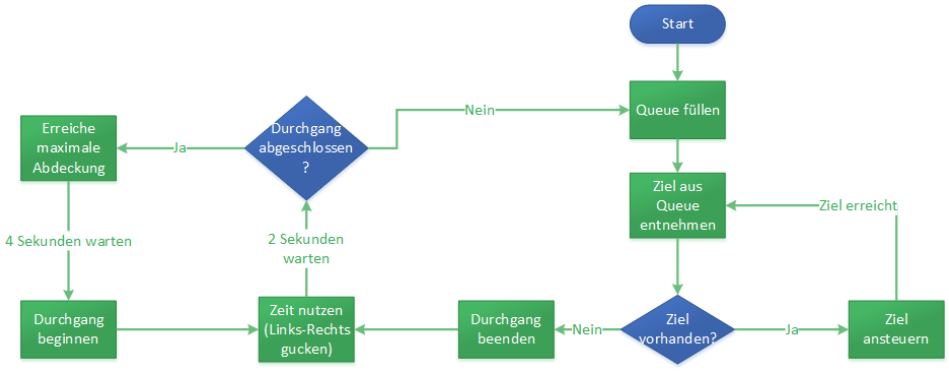


Abbildung 7.1 Der Ansteuerungsverlauf als Flussdiagramm veranschaulicht.

7.3 Erklärung

Durch diesen Zusammenbau hat man viele Freiheiten in der Gestaltung der Kopfsteuerung. So ist es hiermit möglich für Experten der verschiedenen Roboterrollen die Kopfsteuerung durch die Änderung weniger Variablen in dem neuen Modul anzupassen und dabei eine möglichst hohe kooperative Feldabdeckung zu haben. Wie erfolgreich und effizient das Gesamtwerk ist muss bei echten Spielen der SPL evaluiert werden. Um aber einzuschätzen wie gut der Ansatz der Maximierung der kooperativ sichtbaren Fläche ist, welcher das eigentliche Thema dieser Arbeit ist, werden in dem nächsten Kapitel zwei Versuche beschrieben.

Kapitel 8

Evaluation

In diesem Kapitel wird untersucht inwiefern der Ansatz der Maximierung der kooperativ sichtbaren Fläche in der Robotik anwendbar ist. Dazu werden zwei Versuche durchgeführt. Der Erste dient zum Vergleich der Ereigniserkennung und der Zweite überprüft den Einfluss auf die Lokalisierung.
Alle Versuche die auf echten Robotern durchgeführt wurden fanden im Projektraum statt in dem ein Spielfeld mit verringerten Ausmaßen vorhanden ist.

8.1 Ereigniserkennung

In diesem Versuch stehen die Roboter an ihren fest definierten Startpositionen und führen ihre Kopfbewegungen durch. Dabei werden hier die Flächenabdeckung und das Links und Rechts gucken verglichen. Als Ereignis ist hier das herein rollen des Balles aus verschiedenen Positionen definiert. Der Aufbau des Versuchs ist in Abb. 8.1 dargestellt. Dabei bezeichnen die Positionen 1 und 2 die Positionen der Roboter mit der entsprechenden Spielernummer. Diese Positionen entsprechen den Einlaufpositionen des eigenen Torwarts und der des Gegnerischen. Einlaufpositionen für Torwarte sind Randpositionen in der Nähe des eigenen Tors mit Blick auf dessen Mitte. Die Positionen sind aus zwei Gründen so gewählt worden. Zum einen sind die Positionen weit genug von einander entfernt, um den Prozess der Maximierung zu verfolgen und andererseits waren diese Positionen leicht für die Evaluation einzuprogrammieren, somit bedurfte es einiger Änderungen in dem Modul für die Selbstlokalisierung

damit die Tests durchgeführt werden konnten. Die Einwurfpositionen A - M wurden so gewählt, dass durch die überwiegend symmetrische Anordnung eine hohe Chance der Ereigniserkennung existiert. Zusätzlich wurde B so gewählt, dass man die Unterschiede zwischen den beiden Beobachtungsmethoden deutlich merkt.

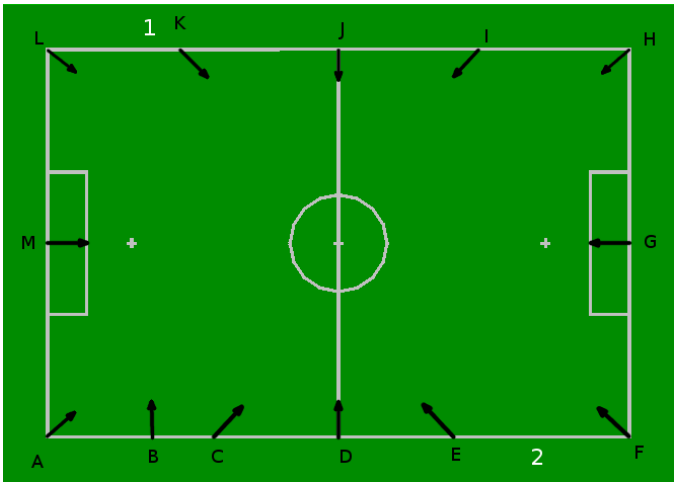


Abbildung 8.1 Der Versuchsaufbau für die Ereigniserkennung.

Das Ergebnis:

Modus	**Erkannte Balleinwürfe**	
	Spieler 1	Spieler 2
Links-Rechts-Steuerung	12	13
Maximierung der Fläche	13	10

Tabelle 8.1 Das Ergebnis der Ereigniserkennung auf echten Robotern.

Dieser Versuch wurde auch in der Simulation mit zwei vollständigen Teams ausgeführt. Dabei hat das blaue Team die Maximierung der kooperativ sichtbaren Fläche und das rote Team die Links-Rechts-Steuerung durchgeführt. Die Spieler hatten dabei ihre Einlaufpositionen inne. Der Versuch wurde 4 simulierte Minuten lang durchgeführt in denen der Ball manuell über etliche Positionen des Feldes geschoben wurde. Als Ergebnis für die Ballerkennungen wurde die Anzahl der validen Ballmodelle des gesamten Teams gemessen. Vergleiche Abb. 8.2.

Das Ergebnis des Simulationsversuchs:

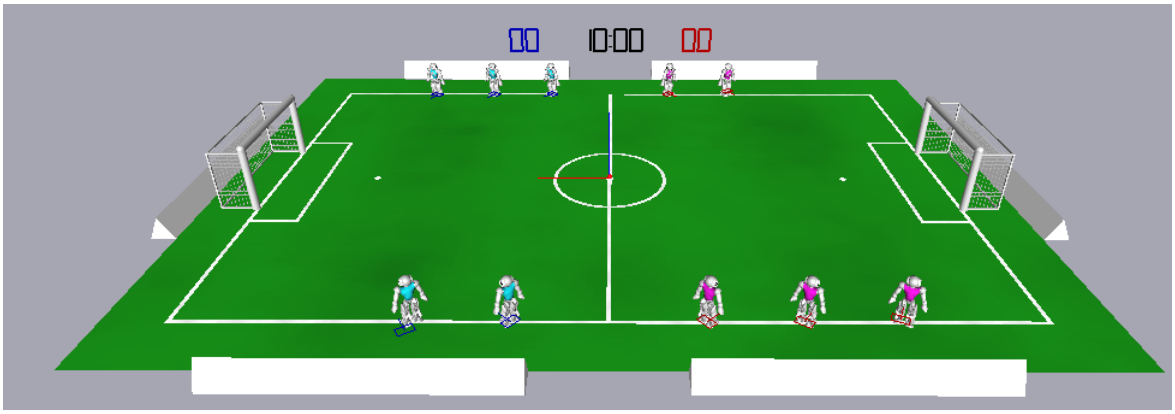


Abbildung 8.2 Aufbau des Simulationsversuchs.

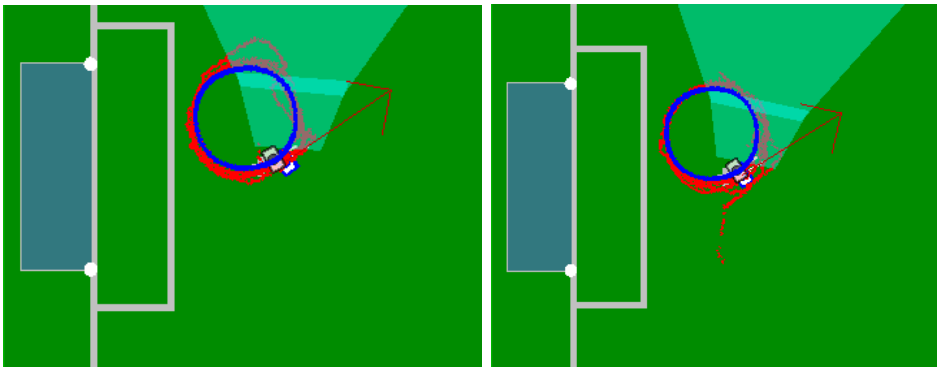
Steuerungsart	Ballerkennungen im Team
Links-Rechts-Steuerung	13118
Maximierung der Fläche	12930

Tabelle 8.2 Das Ergebnis der Ereigniserkennung in der Simulation.

8.2 Lokalisierung

Um die Lokalisierung zu testen wurde dieser Versuch an das Experiment von Seekircher [6] angelehnt. In diesem Versuch soll der der Roboter in einer wiederholbaren Form über das Feld laufen und dabei seinen Lokalisierungsverlauf aufzeichnen.
Der Roboter startet von der Strafstoßmarke aus und läuft mit möglichst konstanter Geschwindigkeit im Kreis während er dabei die ganze Zeit eine Kopfsteuerung ausführt. Jeder Durchgang dauert solange bis der Roboter 5 Runden gelaufen ist. Auch in diesem Versuch wird die Maximierung der kooperativ sichtbaren Fläche mit der Links-Rechts-Kopfsteuerung verglichen. Dabei wurde das Modul für die Lokalisierung in diesem Versuch nur an der Startposition für den Roboter verändert. Die Parameter für die Maximierung wurden insofern angepasst damit die Steuerung und Berechnung schneller erfolgt. Die Wahl der Strafstoßmarke als Startpunkt ist getroffen worden, da der nächste aus Prinzip geeignete Startpunkt der Mittelpunkt wäre. Doch in der Lokalisierung ist eine korrekte Berechnung die in der Mitte beginnt sehr schwierig, da durch die symmetrische Anordnung alle Landmarken

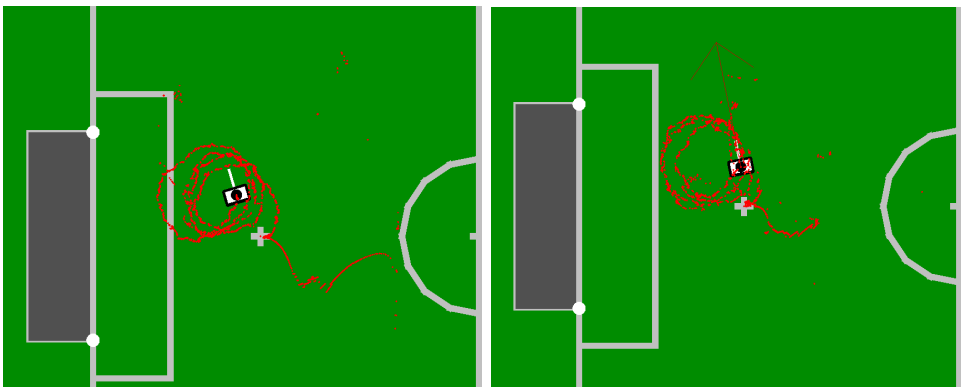
mindestens zweimal vorkommen und somit die gespiegelte Position ebenso wahrscheinlich ist wie die Tatsächliche. Die Strafstoßmarke besitzt damit den Vorteil genau auf einer Hälfte zu sein und der Roboter sollte dann durch das Extrahieren von Informationen aus den Sensordaten seine Position ermitteln können. Als Bewegungsmuster wurde der Kreis gewählt da dieser mit wenig Aufwand und großer Fehlerfreiheit erzeugt werden kann. Komplexere Muster könnten durch die Lokalisierung, die hier getestet werden soll, entarten und somit die Informationsgewinnung stören. Daher ist die Wahl dieser einfachen Form durchaus vernünftig.
Der Versuch wurde wie oben beschrieben auch in der Simulation durchgeführt um etwaige Fehler durch eine ungünstige Farbkalibrierung oder beschädigte Roboter zu vermeiden. In der Abb. 8.3 ist zu sehen wie sich die Lokalisierung bei den verschiedenen Kopfsteuerungen verhält.



(a) Maximierung der sichtbaren Fläche. (b) Links-Rechts-Steuerung.

Abbildung 8.3 Die angewandten Steuerungen in der Simulation, wobei der rote Pfad die ermittelte Lokalisierungsposition und der Blaue die durch die Simulation gelieferte tatsächliche Position des Roboters darstellt.

Derselbe Versuch wurde auch mit einem echten Roboter durchgeführt. Genau wie bei der Simulation durfte der Roboter fünf Runden lang im Kreis laufen und wendete dabei die zu vergleichenden Kopfsteuerungen an. In Abb. 8.4 sieht man den Lokalisierungsverlauf anhand der roten Punkte.



(a) Maximierung der sichtbaren Fläche. (b) Links-Rechts-Steuerung.

Abbildung 8.4 Die aufgezeichnete Lokalisierung (rot) mit den beiden Kopfsteuerungen aus den Logs eines echten Roboters.

Kapitel 9

Fazit und Ausblick

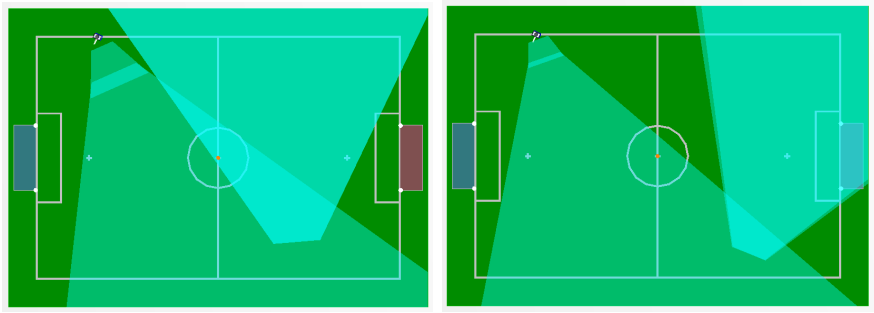
In diesem Kapitel werden die Versuche aus dem Kapitel Evaluation analysiert und die insgesamt erbrachte Leistung in dem Rahmen dieser Arbeit kritisch betrachtet. Anschließend wird ein Ausblick auf eventuelle Änderungen in der Thematik und spezifische Erweiterungsmöglichkeiten dieser Arbeit aufgezeigt. Am Ende des Kapitels wird ein Fazit für diese Arbeit gezogen.

9.1 Auswertung

9.1.1 Ereigniserkennung

Wie an den Werten in der Tabelle 8.1 zu erkennen ist hat der Ansatz der Maximierung bei der Ereigniserkennung keinen besonderen Erfolg gehabt. Beachtlich war aber dennoch der Erfolg der Links-Rechts-Steuerung, da durch die permanente Bewegung eine Verwischung im Bild erreicht wird und dadurch Störungen in der Erkennung vorhanden sein können. Da die Auswertung anhand der von B-Human generierten Logs erfolgte, war sehr gut zu erkennen, dass der Ballerkenner auch bei recht starker Verwischung Bälle erkennen kann. Dieser Effekt der Verwischung wurde vor dem Versuch als wesentlich stärker eingeschätzt und dadurch vermutet, dass dadurch weniger Bälle erkannt werden. Das Problem der Verwischung existiert aber auch bei der Maximierung, da der Ball eine gewisse Geschwindigkeit besitzt. Insgesamt ist aber bei dem Sichten der Logs zu beobachten, dass bei den durch die Maximierung entdeckten Bällen die Erkennung konstanter verlief.

Da durch Störungen, ruckartige Bewegungen und dergleichen die Erkennung erschwert wird kann man nicht erwarten, dass der Ballerkenner über jedes einzelne Bild den Ball erkennt.
Die Ursache für das schlechtere Abschneiden der Maximierung liegt darin begründet, dass die Sichtfelder nicht so begrenzt wie in der Simulation berechnet waren und daher die Optimierung nicht wie geplant von statten ging. Das Sichtfeld des Roboters mit der Nummer 1 war größer als in der Simulation und ließ Roboter Nummer 2 nicht den vollen Handlungsspielraum. Daher kam es auch im Versuch zu ungewollten Nebenwirkungen und dadurch lief die Maximierung nicht wie geplant. Dennoch war in den Tests nicht permanent eine Art der Maximierung zu sehen sondern wechselte gelegentlich, weshalb das Ergebnis nicht so extrem schlecht für die Maximierung ausfiel. Die Veranschaulichung der beiden Maximierungen sind in Abb. 9.1 zu sehen.



(a) Die gewünschte Maximierung. (b) Die erreichte Maximierung.

Abbildung 9.1 Die Maximierungen die während des Tests auftraten. Bild (b) ist nicht absolut korrekt, da es, wie das Erste, in der Simulation erzeugt wurde und nicht aus dem Versuch entnommen wurde.

Eine weitere Erkenntnis, zu der ich in diesem Test gelangte, ist, dass diese Implementierung tatsächlich enorm auf die Lokalisierung angewiesen ist und dort schon die kleinsten Fehler eine vollkommen nutzlose Kopfsteuerung zufolge hat.
In dem Simulationsversuch mit den ganzen Teams wurde auch keine günstige Maximierung erreicht. Da alle Roboter gleichzeitig versuchen die Fläche zu maximieren und dabei sehr nahe beieinander stehen, machen diese oft die gleiche Bewegung. Auf diese Weise entsteht eine ungünstige Abdeckung, wie in Abb. 9.2 gezeigt. Da aber durch die gleichzeitigen Bewegungen immer

noch eine bessere Abdeckung möglich ist, erzeugt die Steuerung immer mehr Bewegungen, weshalb dennoch viele Bälle erkannt werden. Dadurch lässt sich der kleine Unterschied aus der Tabelle 8.2 erklären.

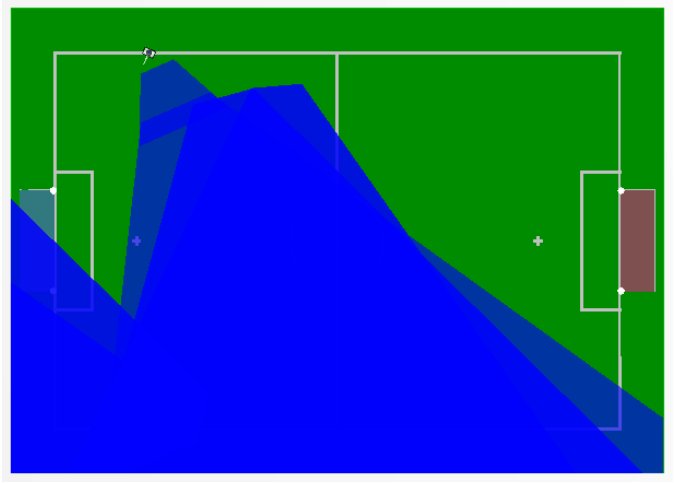


Abbildung 9.2 Eine Abdeckung die von den fünf Robotern im blauen Team erreicht wurde.

9.1.2 Lokalisierung

Im Bereich der Lokalisierung wurde ebenfalls darauf gesetzt, dass die Verwischung durch schnelle Kopfbewegungen stärker ist, als sich bereits in der Ereigniserkennung gezeigt hat. Der Unterschied zwischen den verglichenen Kopfsteuerungen ist in der Simulation, wie in Abb. 8.3 dargestellt, ziemlich gering, da die Simulation für möglichst saubere Bilder sorgt und somit in beiden Verfahren die Lokalisierungen in etwa gleichwertig waren.
Unterschiede die man durchaus während des Versuchs beobachten konnte und teilweise auch in den Bildern sehen kann, sind die weniger sprunghaften Lokalisierungen bei der Maximierung der sichtbaren Fläche. Die Ansammlung der roten Roboterpositionen ergab bei diesem Ansatz eher durchgängige Linien während bei der Links-Rechts-Steuerung die Aktualisierung der Position scheinbar nicht kontinuierlich geschah. Ausreißer in den Lokalisierungen sind in beiden Fällen vorhanden, da existierten nur Unterschiede im zeitlichem Aufkommen. Bei der Maximierung sind diese Ausreißer erst bei der zweiten oder dritten Runde aufgetreten, während bei der Links-Rechts-Steuerung diese Vorfälle ganz am Anfang einmal aufgetreten sind und dann erst wieder

in der vierten Runde.
Insgesamt ist dennoch festzuhalten, dass sich beide Methoden in der Simulation nicht deutlich unterscheiden, auch wenn bei der Links-Rechts-Steuerung unten vereinzelte Punkte sehr weit von der eigentlichen Position entfernt sind. Die Simulation bietet einige Vorteile, da die Bildfehler dort gering sind und auch keine Fehler durch die Hardware oder Kalibrierung vorhanden sind.
Die Versuche auf dem echten Roboter wurden für beide Kopfsteuerungen mehrmals wiederholt, da in den Aufzeichnungen oft Lokalisierungssprünge in einem Ausmaße vorkamen, dass sich die Steuerungen nicht vergleichen ließen. Dieses Phänomen zeigt deutlich warum das Testen auf Robotern so wichtig ist. In der Simulation sind eine perfekte Kalibrierung und saubere Perzepte gewährleistet und deswegen arbeiten die meisten Erkenner in dieser Umgebung vorzüglich, aber sind dafür in der Realität sehr fehleranfällig. Ein Erkenner bei dem diese Beschreibung zutrifft, ist der Erkenner für weiße Tore, der in diesem Jahr innerhalb des Projekts B-Human geschrieben wurde.
Falsch erkannte Tore können die ganze Lokalisierung zunichte machen, da sie eine geeignete Landmarke darstellen und dann in der Lokalisierung verrechnet werden. Dieser Torerkenner setzt darauf weiße Regionen im Bild zu finden und dann die Form eines Torpfostens darin zu erkennen. Die Torerkennung in B-Human musste sich lange Zeit entwickeln. Ungünstig im Sinne dieser Evaluation ist, dass diese Entwicklung in einem separatem Git-Branch geschah und nicht die aktuellste Version des Torerkenners nutzen konnte, die wesentlich zuverlässiger die Tore erkennt. Daher kommen False-Positives wie in der Abb. 9.3 zustande. Dies geschieht häufiger, da im Projektraum die Farbe Weiß recht häufig vorkommt.
Diese Fehlerquelle ist jeder der zwei Kopfsteuerungen gegeben und sollte auch vergleichbare Ergebnisse liefern. Wie in der Abb. 8.4 zu sehen ist, ist auch auf dem echten Roboter kein großer Unterschied zwischen den beiden Kopfsteuerungen vorhanden. Die Lokalisierung macht bei beiden Steuerung direkt am Anfang einen Schlenker der zur Strafstoßmarke führt. Anschließend läuft auch die Lokalisierung die Kreise mehr oder weniger kontinuierlich ab. Wenn man die Pfade miteinander vergleicht sieht man, dass bei der Maximierung der sichtbaren Fläche, der Pfad etwas kontinuierlicher verläuft und die ganzen Punkte minimal näher bei einander liegen. Dennoch

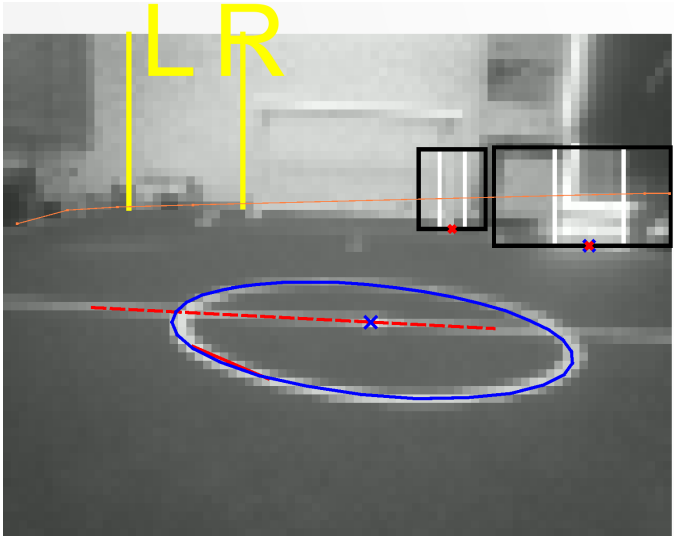


Abbildung 9.3 Ein falsch erkanntes weißes Tor, was zur Lokalisierung verwendet wird, ist mit gelben Linien und einer links(L) und rechts(R) Kennzeichnung eingezeichnet.

haben beide Steuerungen Ausreißer dabei, die sich als Punktewolken oben und rechts in den Bildern äußern. Die Form des Kreises unterscheidet sich in beiden Steuerungen sehr von dem Versuch in der Simulation, da sich in Wirklichkeit der Roboter nicht so gleichmäßig und präzise bewegt wie der simulierte Roboter. Dies kann an eventuellen Schäden an der Hardware liegen, aber auch daran dass die Gelenkmodellierung nicht perfekt ist. Dadurch hat sich bei beiden Versuchen der Kreis im Laufe der Umdrehungen nach Links verlagert, dabei war dies Phänomen bei der Maximierung der sichtbaren Fläche ausgeprägter, da im Laufe der Versuchsreihen der Roboter schon warm gelaufen ist und dadurch die Bewegungen etwas gelitten haben.
So zeigt dieses Experiment, dass obwohl die Maximierung der sichtbaren Fläche von der Lokalisierung stark abhängig ist, diese doch nicht das Lokalisierungsverhalten verschlechtert.

9.2 Ausblick

Im Rahmen der Zielspezifikation in der Kopfsteuerung sind viele Arten der Erweiterung und Anpassung möglich. Zusätzlich werden solche Möglichkeiten über die Rollenspezifikationen unterstützt. Konkrete Erweiterungen, die man

umsetzen könnte, wären gesonderte statische Feldpositionen die für bestimmte Rollen interessant sind. Dabei wäre es sogar möglich solche Punkte zu lernen, wie es in anderen Arbeiten sogar der Fall ist vgl. [7],[9]. Solche Erweiterungen würden neue Zielspezifikationen und erweiterte Rollenspezifikationen benötigen, die eine Ansteuerung zulassen. Diese Arbeit liefert hierfür die geeignete Grundlage, da hier das Verhalten leicht verändert werden kann und neue Ziele leicht zu etablieren sind.

Im Hinblick auf die Ereigniserkennung kann nach der Evaluation gesagt werden, dass die hier entwickelte Kopfsteuerung sehr abhängig von einem korrektem Sichtfeld und einer stabilen Kommunikation ist. Außerdem ist die Wechselwirkung zwischen der Kopfsteuerung und der Lokalisierung kritisch. Man könnte die Abhängigkeiten reduzieren indem man nicht auf die W-Lan-Kommunikation setzt, sondern selber die Daten aus den Sensoren extrahiert und dadurch sogar ein weiter beschränktes Sichtfeld, was Hindernisse und andere Spieler berücksichtigt, erhält. Dies wäre, wenn die Extraktion der Daten gelingt, vor allem wegen der Gaußschen Trapezformel möglich, da Einschränkungen des Sichtfeldes aus dem ursprünglichem Sichtfeld geschnitten werden können, ohne dass die Flächenberechnung angepasst werden muss. Dabei ist zu beachten, dass aber eine präzise Bestimmung nicht mehr erfolgen kann, wenn sich die Sichtfelder der Roboter kreuzen.

Die Maximierung der kooperativ sichtbaren Fläche hat nach der Lokalisierungsevaluation keinen entscheidenden Einfluss auf die Lokalisierung. Obwohl die Hoffnung bestand, dass durch die Vergrößerung des Sichtfeldes mehr Landmarken gesehen werden und durch die Ausführung von weniger hektischen Kopfbewegungen diese Marken zuverlässiger erkannt werden. Dies hat sich in der Evaluation nicht bestätigt und legt nahe für die Lokalisierung eine gesonderte Kopfsteuerung zu nehmen, die gezielt Landmarken sucht und somit Unsicherheiten in der Lokalisierung minimiert. Da sind Lern- und Explorationsansätze wie im Kapitel 3 vorgestellt eher von Erfolg gekrönt, da geometrische Ansätze höchst wahrscheinlich immer stark mit der Lokalisierung wechselwirken.

Der hier vorgestellte Ansatz setzt einige Operationen recht effizient um, dennoch wäre es vorstellbar einige Bereiche noch zu optimieren. Das Scanline-Verfahren ist dazu gedacht schnell Schnittpunkte zu bestimmen und ist gut anzuwenden. Dennoch ist eine weitere Optimierung im Hinblick auf die Überprüfung denkbar, da in diesem Ansatz immer noch zu viele Überprüfungen statt finden und nach dem Prinzip der Vorlesung [15] einige Überprüfungen

gespart werden können, wenn ein Modell existiert, was mit beliebig vielen horizontalen und vertikalen Segmenten arbeiten kann. Ebenso kann man die verwendete Datenstruktur kritisch betrachten und eventuell verbessern. Die hier verwendeten Rot-Schwarz-Bäume wurden gewählt, da sie eine äußerst beliebte Baum-Variante sind und gut aus anerkannten Quellen zu beschaffen sind. Viele ihrer Operationen die im Rahmen der Schnittpunktberechnung entwickelt wurden sind vielleicht nicht auf die effizienteste oder gar eleganteste Weise implementiert, aber dennoch recht effektiv. Es wäre eine Überlegung wert das Scanline-Verfahren auf eine andere effiziente Baum-Variante umzustellen, wie zum Beispiel den AVL-Baum(vgl. [4], S. 284 - 296) und die Laufzeiten zu vergleichen.
Bei der Berechnung der Flächenabdeckung wäre es auch denkbar den Speicherbedarf und an Stellen wie der Zusammenfassung der Fläche den Berechnungsaufwand zu verringern. Wie in dem Kapitel der Berechnung der Fläche erwähnt, ist die hier vorgestellte Methode nur eine grobe Heuristik, da man die Schnittflächen von den beliebigen Schnittpolygonen nicht präzise berechnen kann. Hier könnte man andere Methoden verwenden, die eventuell präzisere Ergebnisse liefern und dabei weniger aufwändig sind. In dieser Hinsicht könnte man zum zusammenfassen dieser Schnitte eine konvexe Hülle bilden, die mit einem Verfahren wie dem Graham's Scan (vgl. [4], S. 475 - 478) effizient berechnet werden kann. Durch ein solches Verfahren wird die Schnittfläche immer etwas überschätzt, aber würde dafür Schnittflächen stärker bestrafen, was einer optimalen Abdeckung sicherlich zuträglich ist.

9.3 Fazit

Das Prinzip der Maximierung der kooperativ sichtbaren Fläche hat sich als geometrisches Prinzip der Kopfsteuerung leider als nicht durchgängig brauchbar herausgestellt. Für die Ereigniserkennung war der Ansatz nicht so effektiv wie gewünscht und die Wechselwirkung mit der Lokalisierung hat sich als kritisch erwiesen. Daher wird eine durchgängige Nutzung ausgeschlossen. Eine Nutzung im Bereich der Ballsuche ist durchaus sinnvoll aber laut Evaluation wäre eine permanente Links-Rechts-Steuerung mit Positionswechseln mindestens ebenso effektiv.
Ziel dieser Arbeit war es die Anwendbarkeit eines geometrischen Ansatzes

zu zeigen. Dies ist hier gelungen. Da die *NAO*s mit einem Intel Atom Prozessor ausgestattet sind, besitzen diese genügend Rechenkapazität, um auch geometrische Ansätze zu verfolgen. Somit zeigt diese Arbeit, dass in der Robotik die Performanz immer noch wichtig ist, aber mittlerweile auch ehemals als undurchführbar geltende Ansätze durchführbar sind.

Literaturverzeichnis

[1] Parker, Lynne E.: Distributed algorithms for multi-robot observation of multiple moving targets. In: *Autonomous robots* 12 (2002), Nr. 3, S. 231–255

[2] Bender, Peter: Eine einfache Formel für den Flächeninhalt von Polygonen. In: *Katja Krüger und Philipp Ullmann (Hg.): Von Geometrie und Geschichte in der Mathematikdidaktik. Festschrift zum* 65 (2012), S. 53–70

[3] Sedgewick, Robert: *Algorithmen in C++, Teile 1- 4, Grundlagen, Datenstrukturen, Sortieren, Suchen.* PEARSON EDUCATION DEUTSCHLAND, 2002. – ISBN 3–8273–7026–4

[4] Ottmann, Thomas ; Widmayer, Peter: *Algorithmen und Datenstrukturen.* Spektrum Akademischer Verlag, 2012. – ISBN 978–3–8274–2803–5

[5] Burger, Wilhelm ; Burge, Mark J.: *Digitale Bildverarbeitung - Eine Einführung mit Java und ImageJ.* Springer, 2006. – ISBN 3–540–21465–8

[6] Seekircher, Andreas ; Laue, Tim ; Röfer, Thomas: Entropy-based active vision for a humanoid soccer robot. In: *RoboCup 2010: Robot Soccer World Cup XIV.* Springer, 2011, S. 1–12

[7] Guerrero, Pablo ; Solar, Javier Ruiz-del ; Romero, Miguel: Explicitly task oriented probabilistic active vision for a mobile robot. In: *RoboCup 2008: Robot Soccer World Cup XII.* Springer, 2009, S. 85–96

[8] Seara, Javier F. ; Strobl, Klaus H. ; Schmidt, Günther: Information management for gaze control in vision guided biped walking. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on* Bd. 1 IEEE, 2002, S. 31–36

[9] Kwok, Cody ; Fox, Dieter: Reinforcement learning for sensing strategies. In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on* Bd. 4 IEEE, 2004, S. 3158–3163

[10] Röfer, Thomas ; Laue, Tim ; Müller, Judith ; Fabisch, Alexander ; Feldpausch, Fynn ; Gillmann, Katharina ; Graf, Colin ; Haas, Thijs J. ; Härtl, Alexander ; Humann, Arne ; Honsel, Daniel ; Kastner, Philipp ; Kastner, Tobias ; Könemann, Carsten ; Markowsky, Benjamin ; Riemann, Ole Jan L. ; Wenk, Felix: *B-Human Team Report and Code Release 2011.* 2011. – Only available online: `https://www.b-human.de/downloads/bhuman11_coderelease.pdf`

[11] Röfer, Thomas ; Laue, Tim ; Müller, Judith ; Bartsch, Michel ; Batram, Malte J. ; Böckmann, Arne ; Böschen, Martin ; Kroker, Martin ; Maass, Florian ; Münder, Thomas ; Steinbeck, Marcel ; Stolpmann, Andreas ; Taddiken, Simon ; Tsogias, Alexis ; Wenk, Felix: *B-Human Team Report and Code Release 2013.* 2013. – Only available online: `https://github.com/bhuman/BHumanCodeRelease/tree/coderelease2013`

[12] RoboCup Technical Committee: *RoboCup Standard Platform League (NAO) Rule Book.* 2015. – `http://www.informatik.uni-bremen.de/spl/pub/Website/Downloads/Rules2015.pdf` Abgerufen am: 24.08.2015

[13] The Robocup Federation: *RoboCup.* 2015. – `http://www.robocup.org/about-robocup/` Abgerufen am: 23.08.2015

[14] The Robocup Federation: *Soccer Standard Platform League.* 2015. – `http://www.robocup.org/robocup-soccer/standard-platform/` Abgerufen am: 23.08.2015

[15] Leitte, Heike: *Algorithmische Geometrie - 3. Schnitte von Liniensegmenten.* 2014. – `http://www.iwr.uni-heidelberg.de/`

groups/CoVis/Teaching/AG_SS14/AG_3_LineSegmentIntersection.pdf Abgerufen am: 26.08.2015

[16] Aldebaran Robotics: *More about NAO*. 2015. – https://www.aldebaran.com/en/more-about Abgerufen am: 18.10.2015

[17] Sedonia Technologies: *NAO_DEBOUT*. 2015. – http://www.sedoniatech.com.au/imgs/nao_debout.jpg Abgerufen am: 23.08.2015

[18] Aldebaran Robotics: *NAO - Video camera*. 2015. – http://doc.aldebaran.com/2-1/family/robots/video_robot.html Abgerufen am: 23.08.2015

[19] Team B-Human: *B-Human, Willkommens-Seite*. 2015. – https://www.b-human.de/?lang=de Abgerufen am: 23.08.2015

[20] Eigen: *Linear algebra and decompositions*. 2015. – http://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html Abgerufen am: 28.08.2015